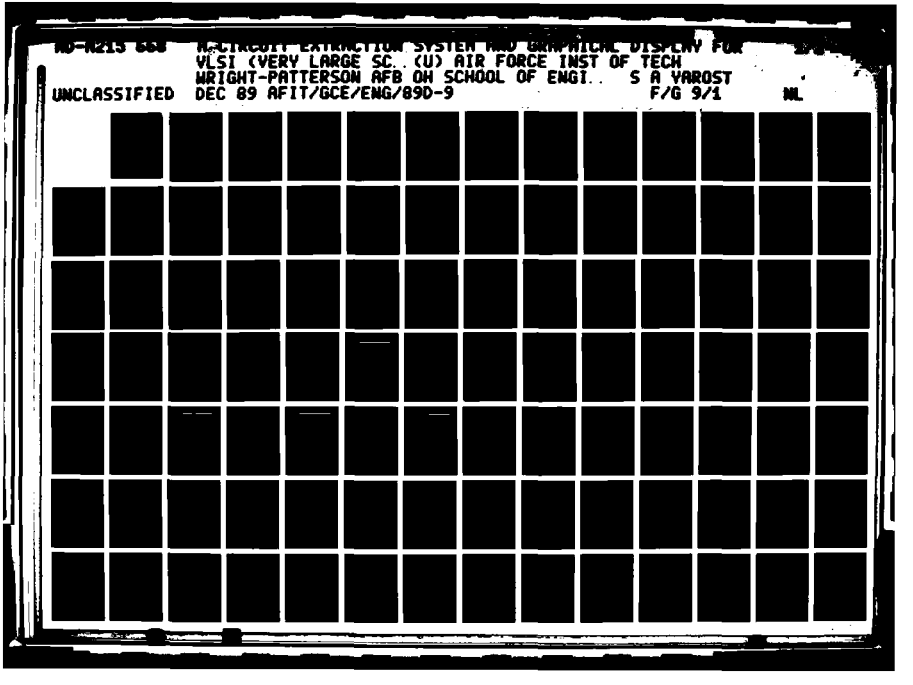
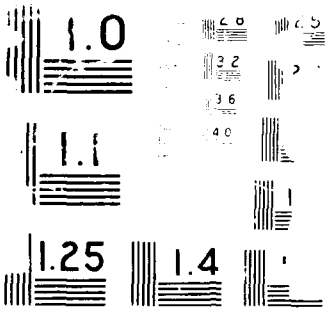

AD-A215 868
VLSI (VERY LARGE SC..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S A YAROST
UNCLASSIFIED DEC 89 AFIT/GCE/ENG/89D-9 F/G 9/1 NL


1.0
1.1
1.25
1.4

A CIRCUIT EXTRACTION SYSTEM AND GRAPHICAL DISPLAY FOR VLSI DESIGN

THESIS

Stuart A. Yarost
Captain, USAF
AFIT/GCE/ENG/89D-9

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/89D-9

A CIRCUIT EXTRACTION SYSTEM AND GRAPHICAL DISPLAY FOR VLSI DESIGN

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

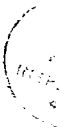
Stuart A. Yarost, B.S.
Captain, USAF

December 1989

Acknowledgments

I would like to express my appreciation to my wife, Debbie, for her love and patience while I spent all my time communing with my computer.

I wish to thank Doctor Frank Brown for his guidance, encouragement and help when this thesis looked like a never ending project, as well as Maj Joe Degroat for his enthusiasm and ideas. I also would like thank CPT Bob Hammell for his correction of my drafts, as well as LTC Charles Bisbee and Capt Bruce George for their encouragement. I have a special thanks for Tony Schooler, whose help with the graphics on the Sun 3/50s allowed me to complete my thesis.



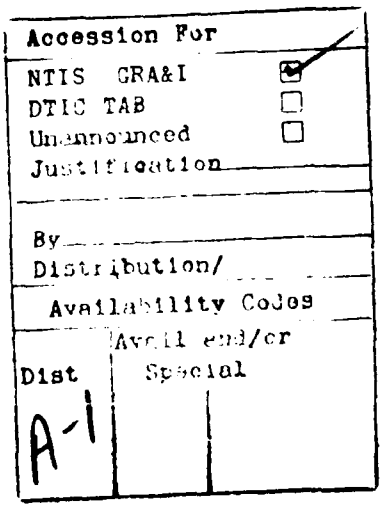


Table of Contents

List of Figures

Abstract

This thesis proposes a system for higher-order logic extraction of components from a net-list of transistors and the graphical display of the extracted components. Critical sections have been implemented to demonstrate the feasibility of the system. These sections include a prototype expert system written in CLIPS and a graphical display capable of displaying extracted components on a Sun workstation.

Extraction techniques which were developed in this effort use pattern matching and multiple passes. Graphical techniques used in the display include simple line drawing and translation of images.

This research has the potential to provide savings of time and effort to engineers designing new circuits or reverse-engineering older circuits for which no adequate specifications exist. This system will also help to close the design cycle and allow the designer to assure that what he has physically designed is what he logically designed.

A CIRCUIT EXTRACTION SYSTEM AND GRAPHICAL DISPLAY FOR VLSI DESIGN

I. Introduction

Background

The design of a very large scale integrated (VLSI) circuit can take man-years, and can cost millions of dollars. Engineers are searching for ways to reduce the amount of time and the amount of money needed to design such circuits. Computer Aided Design (CAD) tools reduce the time needed.

Another task that takes many man-hours is the redesign of existing VLSI circuits whose original documentation is either missing or inaccurate. Reverse-engineering is the taking apart of an existing circuit to find how it is put together. This task is much harder than the original design of a circuit.

The tasks of designing a circuit and reverse-engineering a circuit have many common problems. In both cases, extracting the logic of the circuit from the silicon layout and being able to display the logic of the circuit would help facilitate the process. Finding ways to make these tasks easier is a current area of research at many institutes of higher learning, and in many companies.

The CAD cycle starts with the design of a circuit, and finishes with the production of the circuit. The closing of this cycle, to verify that a circuit is what it was designed to be, is called verification. At the Air Force Institute of Technology (AFIT), a number of theses have been written in the last few years dealing with the CAD cycle and the reverse engineering of circuits. CPT Erik Fretheim, USA, did research on the reverse engineering of VLSI circuits using pattern recognition techniques to identify the different components on a silicon chip (1:1-103). CPT Mike Dukes, USA, used the PROLOG language to extract the circuit logic of a chip (2:1-154). Both suggested further research.

The CAD cycle at AFIT needs a number of additional tools to add to the capabilities of those doing circuit design. The same tools can aid the reverse engineering of existing VLSI circuits. This thesis documents the production of such a tool, a graphical display which shows the logic of a circuit using standard logic and gate symbols. This tool uses the coordinates from MAGIC to locate the components in the display, and to locate the part of the circuit to view. The current CAD cycle at AFIT includes the design of circuits in the VHSIC Hardware Description Language (VHDL), their layout using MAGIC and their simulation at the logic-gate level using ESIM.

VHDL is used to completely specify a circuit, and to

simulate it to confirm that the circuit designed is what is needed. VHDL simulations of a circuit can include simulations from the perspective of timing, logic, connections, and components. Once the circuit has been designed, that design must be converted to components on a silicon chip. MAGIC is the CAD tool used to lay out the physical structure of the circuit. CIF can be used to convert the MAGIC file to "cif" (Caltech Intermediate Format), which details the actual physical structure of the chip. ESIM is used to simulate the logic operation of the circuit that has been laid out using MAGIC.

The VHDL circuit description must be converted to MAGIC manually. A MAGIC file is converted to "cif" format by the program CIF called from within MAGIC. The "cif" file can be converted to an ESIM "sim" file using the program MEXTRA. The normal flow is VHDL to MAGIC to CIF to ESIM to MAGIC until the circuit performs correctly. Figure 1 displays the current CAD cycle at AFIT, with the addition of the graphical display produced for this thesis completing the cycle back to the original logic design.

VHDL does not check a physical layout of a circuit, only the interconnections and logic structure. Since VHDL is used to simulate an actual circuit design, the task is to insure that the MAGIC circuit duplicates the interconnections that were simulated in VHDL. This problem, as well as the problem

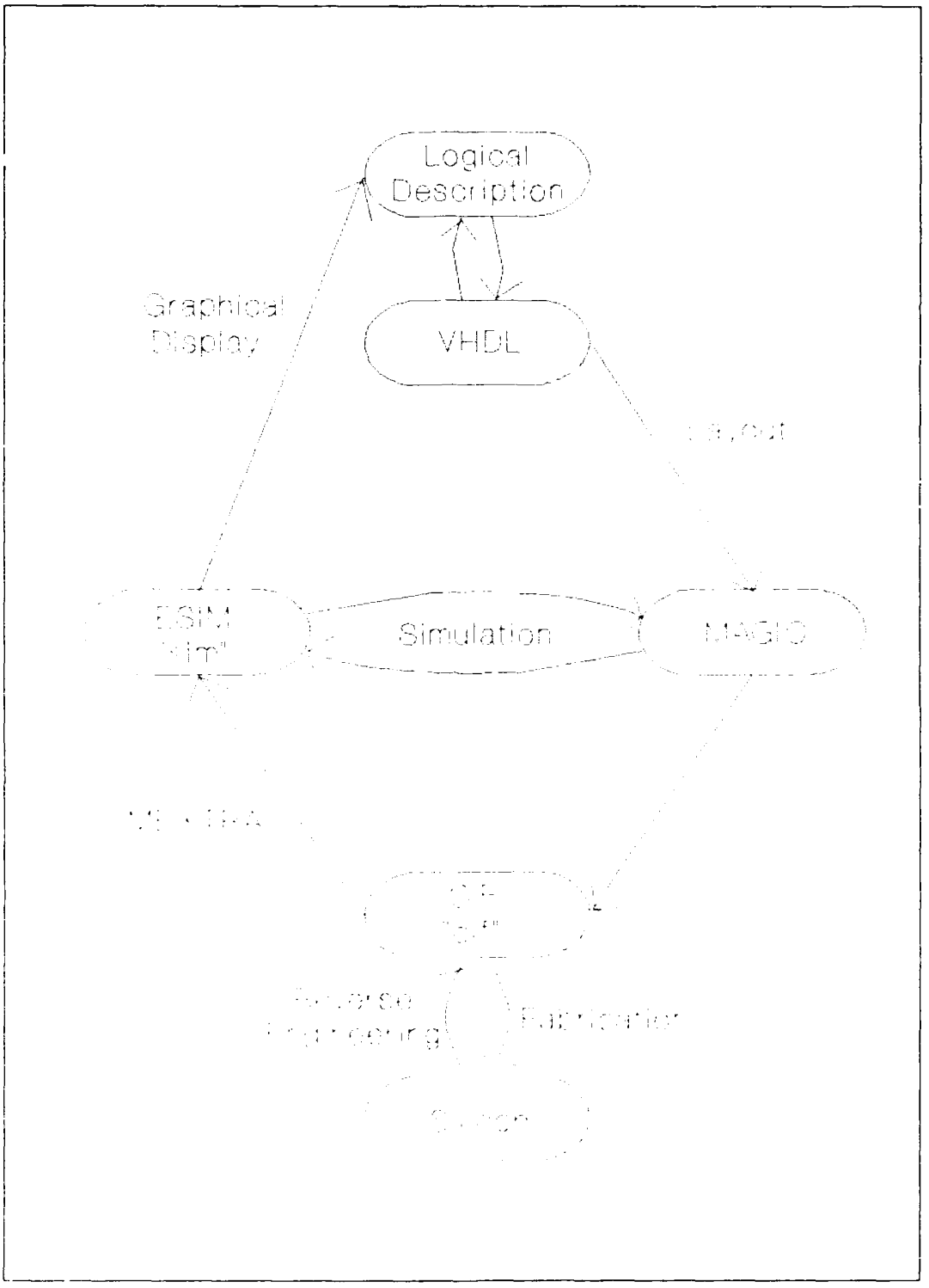


Figure 1. AFIT CAD cycle with Graphical Display

of displaying the logic of an unknown circuit, are problems that need solutions.

Problem

This thesis was undertaken to help solve some specific problems. The overall task was to add some quick, visual feedback to the AFIT design cycle. A graphical display helps the engineer to check if the design of the circuit he laid out has the correct logic. This disply is what has been built. It also helps determine the function of an unidentified circuit, as well as that of a circuit for which no documentation exists. A graphical display is a method to display and manipulate information in a graphical manner, as contrasted with a tabular manner. Instead of listing the components and their gates, a graphical display will show a pictorial representation of each component and its respective gates using standard logic and electrical engineering symbols. A visual representation in this manner is typically easier to understand than a tabular listing of components.

A graphical display was built, using a symbolic form for data storage. The symbolic form was used to allow the manipulation of the net-list by the symbolic language. A symbolic form is a form that a symbolic language, such as PROLOG, can understand. Each symbolic language has its own requirements that need to be satisfied by a symbolic form that

t can read.

To create the graphical display, an expert system was written to determine the logic of a circuit from a list of component transistors of a circuit (net-list). This expert system was built directly on the work of CPT Mike Dukes, USA (2:1-154). The major difference between the extraction routines CPT Dukes created and the one used for the front-end of the graphical display is the symbolic language used. CPT Dukes used PROLOG, a backward-chaining language, while the expert system for the graphical display was written in CLIPS ("C" Language Integrated Production System), a forward-chaining rule-based language. Extensions to CPT Duke's work include the addition of position information and scaling information.

After the expert system was written, a graphical display was written using the Sun Core graphics commands on a Sun 3 workstation. The language that was used to read the symbolic files produced by the expert system, and to drive the graphics routines, was C.

This thesis provides a two-part solution for the aforementioned problems. First, the higher-order logic components are extracted using the forward-chaining expert system. Then the output of the expert system is used to create a graphical representation of the extracted components, in positions related to their positions on the circuit. The

solution to the second part of the problem relied on the completion of the solution to the first part.

Summary of Current Knowledge

At present, the representation of a circuit in symbolic form has been achieved by CPT Dukes (2:33-99). The symbolic language he used was PROLOG, a backward-chaining AI (Artificial Intelligence) language. Using PROLOG, he was able to start with a list of transistors and finish with a list of higher-order components and logic gates.

CPT Dukes started with a MAGIC "cif" file of a circuit. He then used the program MEXTRA to obtain a "sim" file. In this conversion, all the hierarchical information that is present in the original "cif" file is lost. After running a LEX program "upper" to convert all letters to uppercase, he converted the file to a format that PROLOG could load using "sim2pro", a program he wrote in C. He used a PROLOG program called TRANS to extract a higher order logic-gate description of his circuit. The higher-order logic description is in tabular form.

CPT Dukes used the file created by TRANS to verify that the circuit laid out was the circuit designed in VHDL. He is using this technique in further research into formal verification.

I have found no specific reference in the literature on

converting a symbolic representation of a circuit to a graphical representation. Although there is a plethora of information on how to do graphics and routing, that information has to be integrated to achieve the desired results.

Scope

The scope of this project was to provide an expert system program which extracts a higher-order logic representation from a net-list, and a graphical display capable of displaying that representation on a graphics device. The graphical representation shows the circuit by using files produced by the expert system. The display allows choice of level of representation to be viewed (hierarchy), and area of circuit displayed (zooming).

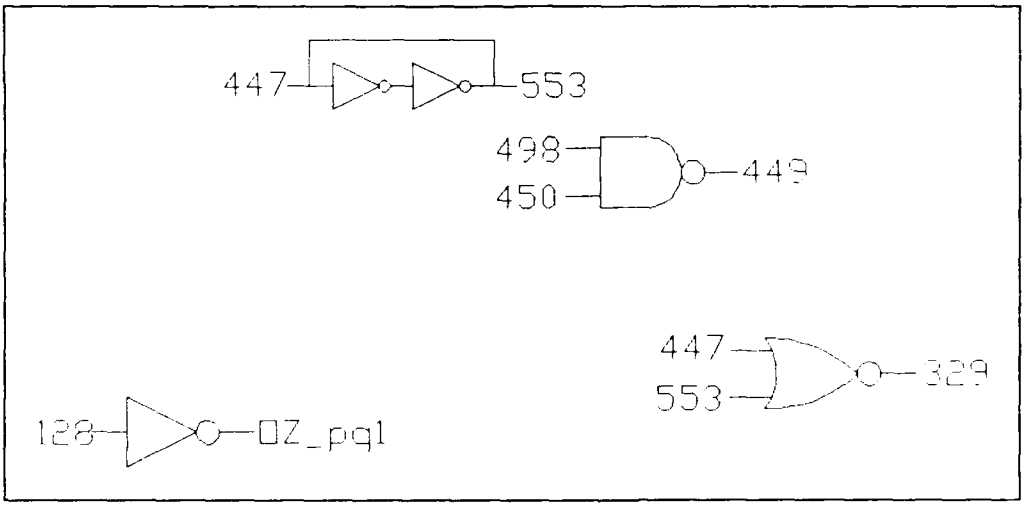
The first part of Figure 2 contains a sample file that has already been parsed to the form that CLIPS can read using a program written for the thesis,"sim2clip". The file contains only n and p type transistors. The second part of Figure 2 shows a file of the components that have been extracted from the first file. These components are a higher order representation of the circuit composed of the transistors in the first part of figure 2. The last part shows what the graphical display would look like for that circuit, using the higher level components.

```
(p 128 OZ_pq1 vdd 300 21596.3 -26700 -24000)
(n 450 449 520 300 450 3300 -1350)
(n 498 520 gnd 300 450 2700 -1350)
(n 453 gnd 329 1200 450 25500 -2250)
(n 447 329 gnd 1200 450 23550 -2250)
(n 584 gnd 533 300 13497.8 -5550 9000)
(p 447 584 vdd 300 6748.75 6150 4200)
(p 584 vdd 533 300 13647.8 -5550 4050)
(p 453 329 589 1200 900 24750 1200)
(p 447 589 vdd 1200 900 23250 1200)
(p 498 449 vdd 300 900 2700 900)
(p 450 vdd 449 300 900 3900 900)
(n 447 gnd 584 300 3299.75 6150 10050)
(n 128 gnd OZ_pq1 300 17996.8 -29700 14400)
```

CLIPS file of unextracted transistors
EXTRACTS TO

```
(buffer 447 533 300 6825)
(inv 128 OZ_pq1 -28200 -4800)
(nand2 gen15 498 450 449 3150 -225)
(nor2 gen16 447 453 329 24262.5 -525)
```

Extracted Components
VISUALIZED AS



Graphical display

Figure 2. Graphical Display Sequence of Events

Approach

The graphical display breaks down into four distinct areas: representing the circuit data symbolically, extracting the higher order logic components, locating their correct relative location on the display, and actually drawing the image on the screen.

PROLOG is not the ideal choice to represent the circuit components, nor to manipulate them to obtain the higher order logic components. A forward-chaining AI language can manipulate the symbolic representation with less programming, and is easier to understand and maintain. CPT Dukes's code was, therefore, converted from PROLOG into CLIPS (a forward-chainer), adding the ability to pass along position information. His C program "sim2pro" was altered to parse the "sim" file into a symbolic form that CLIPS needs at the outset (2:28-30).

Locating the correct relative positions for the extracted gates required the use of a simple mathematical routine. The routine averages separately all the x and y locations of the component gates or transistors. The new x and y is the location of the new gate. This is done using CLIPS, allowing different scaling factors to be used, to allow presentation on different graphics devices.

The coding to produce the graphics took relatively little time once the graphics device and graphics programs were

determined. Depending on the capabilities needed, the Sun 3/50 and Sun 3/60 workstations have three different graphics packages available. The choices are Pixwinn, Sun CGI, and Sun Core. All three packages were able to support the graphical display. Sun Core is the most limited in interfacing with Sunwindows, but is also the easiest to understand and to use for drawing. The built-in graphics on the Sun workstations were able to handle the graphical display, when Sun Core graphics were used.

Sequence of Presentation

Chapter 1 has provided some background on the AFIT CAD environment, and the need for a graphical display. The problem was explained, as well as the scope of the problem and solution. An approach to the problem was summarized.

Chapter 2 is an explanation on how the extraction process works. The goal is explained, as well as the procedure. The relative merits of forward and backward-chaining are also discussed. The problems that occurred are explained, as well as the results.

Chapter 3 is a discussion of the graphical display portion of the thesis. First is an explanation of the goal. Next, the procedures followed to achieve that goal are outlined. Problems encountered are then discussed, followed by a summary of results.

Chapter 4 details the extraction and display of an example circuit, a clock generator chip. The quality of the extraction and display are discussed.

Chapter 5 presents overall conclusions and recommendations. Included are possible improvements to the extraction program and to the program that displays the graphical display.

Appendix A contains all the extraction code, as well as other code needed to facilitate the extraction.

Appendix B contains all the code pertaining to the graphical display.

Appendix C contains all the test code as well as the "sim" file and CLIPS file of the clock generator circuit used as an example in chapter IV.

Appendix D is a user's manual for the extraction routines, and the graphical display.

II. Circuit Extraction

Introduction

This chapter discusses the first part of the process of creating a graphical display. It covers how the extraction of higher-order components from lower-order components was achieved. The sections of the chapter are the goals aimed for, procedures for obtaining a suitable file to start with, converting that file to a usable form, and the extraction process. This chapter also covers the difference between forward-chaining and back-chaining, problems encountered while creating the extraction routines, and the results of the extraction portion of this effort. A users guide on how to do the extraction process is included as part of appendix D.

Goal

The creation of a graphical display to help the reverse engineering of VLSI circuits can be divided logically into two parts: (a) representing a circuit symbolically and extracting the higher-order logic gates from the circuit, and (b) displaying the symbolic information on a graphics-capable device. This separation allows completion of either part independently, since neither part is dependent on the other part except for the data interface. The only knowledge needed by both parts is the exact format of the file that is the

product of the extraction process, and that is read by the program that creates the graphical display.

For purposes of this thesis, representing a circuit symbolically is the conversion of the information that describes the circuit into a form that a symbolic language (such as PROLOG or CLIPS) can use. This symbolic form of the list of transistors (net-list) can then be manipulated by the symbolic language.

Extraction is the process of replacing several components of a given level of complexity with a single higher-order component that performs the same logical function. This higher-order representation is easier for the engineer to understand.

Representing a circuit in a symbolic form and extracting the higher-order logic gates was the first part of the research completed. A file containing higher-order logic constructs, derived from a list of N and P type transistors was produced. The problem divided into three smaller tasks.

The first task was to obtain a net-list of the transistors in a circuit in a form that could be understood by an engineer; the second task was translating that list into a form that CLIPS could understand; and the third task was using CLIPS to extract the higher-order logic constructs from the net-list, leaving a file of the components for further processing, extraction, or display.

Obtaining a Net-List

As stated earlier, the first task in the process of extracting higher-order components from lower-order ones is to get a net-list of the transistors. Obtaining a suitable net-list is not a difficult task. MAGIC can produce a file of a layout in "cif" format. This "cif" file does not have the circuit information in a form that is understandable as a net-list. The file contains information on the different physical layers of a polysilicon chip, not how these layers combine to form transistors, capacitors and other components.

The "cif" description is carried out in a hierarchical manner: when a section of a circuit, a cell, is described only once, then the cell name is repeated wherever else that cell exists in the circuit without repeating the information contained in the original cell description. Cells can be recursively described, using cells within cells. Hierarchical information reduces the amount of information needed to completely describe a circuit

The "cif" file, run through MEXTRA, is converted to a form that contains a readable net-list. This form is the one that the simulation program ESIM can use to run a simulation of the operation of a circuit. MEXTRA translates the information on the physical layout of the circuit into a list of transistors, resistors, capacitors and other types of components. It also contains the size, location, and

interconnections between the components. All hierarchical information has been lost, however. The information on the sizes and locations of the components, and interconnections between the components, is what is needed to do the circuit extraction, but it is in a format CLIPS cannot read. Figure 3 shows a sample of the file that is produced by MEXTRA.

```
TYPE    NODES          WIDTHS    COORDINATES
   p 128 OZ_pq1 Vdd 300 21596.3 -26700 -24000
   p 53 OZ_pq2 Vdd 300 21596.3 73200 -29400
   p 277 Vdd 233 300 13647.8 24450 -9900
   p 329 277 Vdd 300 6748.75 18600 -11400
   p 53 Vdd OZ_pq2 300 21596.3 53100 -17850
   p 233 53 Vdd 300 21596.3 39000 -17100
   p 277 Vdd 233 300 13647.8 24450 -12750
   e 450 449 520 300 450 3300 -1350
   e 498 520 GND 300 450 2700 -1350
   e IZ_go GND 498 300 450 1500 -1350
   e 424 453 GND 300 450 27600 -1650
   e 453 GND 329 1200 450 25500 -2250
```

Figure 3. Net-List used by ESIM.

The first field indicates what type of component is listed in that line. A "p" indicates a p-type transistor while an "e" indicates an n-type transistor. The only components used for extraction are n and p type transistors. Symbols for other types of components (such as resistors or capacitors) can also appear, but are ignored for the purpose of extraction. The next three fields are the node names of gate, source, and drain of the transistor. A node name can be any alphanumeric string. Node names are assigned by MAGIC

if not explicitly given by the designer using MAGIC. The next two fields are the x width and y width of the transistor. They are not used in the extraction process. The last two fields are the x and y coordinates of the transistor, respectively. The widths and the coordinates must all be real numbers.

Using the first line of Figure 3 as an example, the fields have the following meaning. "p" indicates a p type transistor. "128", "OZ_pq1" and "Vdd" are the gate, drain and source, respectively. "300" and "21596.3" are the x and y widths, which are not used in extraction, and "-26700" and "-24000" are the x and y coordinates.

Converting the Net-List to CLIPS Symbolic Form

The next task, getting the file in a form readable by CLIPS, was accomplished by using a parsing program, "sim2clip" (see appendix A), written in C. It was converted from the program "sim2pro", written by CPT Mike Dukes (2:127-131), and altered to parse to CLIPS symbolic form instead of PROLOG symbolic form. This CLIPS form can be read by CLIPS using the load-facts predicate of the CLIPS language. Figure 4 shows the file of Figure 3 after its conversion by "sim2clip".

The fields of the symbolic file produced by "sim2clip" correspond one-to-one with the fields of the file produced by MEXTRA.

```
(p 128 OZ_pq1 vdd 300 21596.3 -26700 -24000)
(p 53 OZ_pq2 vdd 300 21596.3 73200 -29400)
(p 277 vdd 233 300 13647.8 24450 -9900)
(p 329 277 vdd 300 6748.75 18600 -11400)
(p 53 vdd OZ_pq2 300 21596.3 53100 -17850)
(p 233 53 vdd 300 21596.3 39000 -17100)
(p 277 vdd 233 300 13647.8 24450 -12750)
(n 450 449 520 300 450 3300 -1350)
(n 498 520 gnd 300 450 2700 -1350)
(n IZ_go gnd 498 300 450 1500 -1350)
(n 424 453 gnd 300 450 27600 -1650)
(n 453 gnd 329 1200 450 25500 -2250)
```

Figure 4. CLIPS Symbolic Form

Legal values for the fields are the same, with one exception: "p" and "n" are the only legal values for the first field. "p"'s pass unchanged, "e"'s are changed to "n"'s, and any lines that begin with any other symbol are removed. The only changes are the addition of the parenthesis at the beginning and end of each line, and the substitution of a "n" for any "e" in the first field, which indicates the type of component. The only changes necessary to allow CLIPS to read the file using the "load-facts" predicate (3:59) is the addition of the parentheses at the beginning and end of each line. The removal of any line not representing a transistor is for ease of extraction.

Extracting Higher-Order Components

Extracting the higher-order logic components can begin once the net-list is in a usable form. The algorithm used is

relatively straightforward. First the file produced by "sim2clip", containing all the transistors, is loaded into memory. Then groups of components that can be combined to form higher-order components are found, pattern-matching the transistors to form higher-order logic components. The facts denoting the used transistors are retracted, while the newly extracted gates are written to a new file.

The name of this new file is of the form "outcompX.clp", where "X" is the extraction level. The components that remain at the end of the extraction are put in a file similarly called "compremX.clp", where "X" has the same meaning.

The file to which the newly extracted components are written can be used later in the same manner, to form even higher-order logic components. Each iteration of this process reduces the size of the database of components used for extraction. As each new component is extracted, fewer components are left to pattern-match against, reducing the memory and time required to do each succeeding match.

Figure 5 shows a portion of a file of extracted components. The first field identifies the component-type. The last two fields are x and y coordinates, respectively. The rest of the fields are node names, except when a unique identifier ("genxx", where xx is a number) is added as the second field. The use of a unique identifier is explained later.

```
(inv 53 OZ_pq2 74850 -4425)
(inv 233 53 41325 -11325)
(inv 277 233 24375 -8775)
(nand2 gen44 498 450 449 3150 -225)
(inv IZ_go 498 1500 -225)
(inv 424 453 27450 0)
(nor2 gen45 447 453 329 24262.5 -525)
(nor2 gen46 329 424 447 20475 -675)
(inv 450 424 17025 -675)
(inv 449 443 5175 -900)
(inv 177 444 10575 -1050)
(inv 444 450 14925 -1050)
(inv 443 177 7275 -1050)
(inv 128 OZ_pq1 -28350 -4200)
(inv 584 533 -5550 5100)
(inv 447 584 6150 7125)
(inv 533 128 -12825 5025)
```

Figure 5. Logic Components.

The fields have the following meaning in the first line of figure 5: "inv" stands for inverter, "53" and "OZ_pq2" are the input and output of the inverter, and "74850" and "-4425" are the x and y coordinates. The legal values for the node names and the x and y coordinates are the same as in the previous files. The legal values for the first field, the component names, and their corresponding values, are listed in figure 6.

As mentioned previously, the exception to what the fields represent is if the second field starts with "gen". In that instance, the second field is a unique identifier used for the extraction process. All components that have any interchangeable inputs or outputs need this identifier. This unique identifier is added to a component listing during the

ntrans	n-diffusion transistor
ptrans	p-diffusion transistor
inv	inverter
tgate	t-gate
nand2	2 input NAND-gate
nor2	2 input NOR-gate
clk_inv	clocked inverter
buffer	buffer
mux	multiplexor
xnor	exclusive NOR gate
xor	exclusive OR gate
dff	D flip-flop

Figure 6. Legal Component Names

extraction process.

The reason the unique identifier is needed is the lack of a back-tracking facility in CLIPS, and the need to be able to bind a component to a variable to be able to retract it. Without back-tracking, each permutation of the order of the interchangeable gates must be checked explicitly. For the example of a NAND-gate with 3 inputs, A, B and C, the component needs to be checked for matching for further extraction with the input gates in 6 different orders: ABC, ACB, BAC, BCA, CAB and CBA.

The unique identifier allows CLIPS to try to pattern-match a component with interchangeable gates with the gates in all possible permutations. In all cases, the component will have the same unique identifier. Figure 7 shows the CLIPS code to extract an inverter, and the need for a unique identifier.

Figure 7 shows how, for the extraction of an inverter from n and p type transistors, each transistor had to be checked with its source and drain interchanged. The unique ID allows the transistor-fact to be retracted, no matter which configuration of source and drain is correct.

```
(defrule inverter
  (or (ptrans ?id1 ?gate vdd ?a ?x1 ?y1)
      (ptrans ?id1 ?gate ?a vdd ?x1 ?y1))
  (or (ntrans ?id2 ?gate gnd ?a ?x2 ?y2)
      (ntrans ?id2 ?gate ?a gnd ?x2 ?y2))
  ?p <- (ptrans ?id1 $?)
  ?n <- (ntrans ?id2 $?)
=>
  (bind ?xa (/ (+ ?x1 ?x2) 2))
  (bind ?ya (/ (+ ?y1 ?y2) 2))
  (retract ?p ?n)
  (fprintout component
    "(inv "?gate" "?a" "?xa" "?ya")"crlf))
```

Figure 7. CLIPS Code to Extract Inverter.

The reduction of the fact base by retracting the components used to build extracted higher-order components allows parallelization of the problem. Different portions of the circuit can be run through the extraction process separately, then the remaining lower-order components not already used in an extraction can be combined to catch any remaining higher-order components that were not extracted before.

The CLIPS rules to extract higher-order components can be combined in several ways. The underlying principle is that

components that are extracted by a rule in a file should not be used for further extraction by another rule in the same file. Following that principle, all rules that extract higher-order components from the same type of lower-order components, can be grouped together in a single file. The rules for different components do not have to be combined in a single file, however. The extraction rules could be separated into a single file for each component, at the extreme. More rules to be checked against in a single file means slower running of the program, and greater memory usage. Fewer rules in a single file therefore means faster execution, and less memory usage. The complete code to do the extraction is in multiple files, grouped by which lower order components are needed (see appendix A).

Forward-Chaining vs. Backward-Chaining

There are inherent advantages and disadvantages in using either forward or backward-chaining to extract higher-order logic gates from a net-list. Both methods allow the parallelization of the problem, as explained in the previous section, as well as the ability to do multiple passes to reduce the complexity of the problem. CPT Mike Dukes used PROLOG (an inherent backward-chainer) to do his extraction (2:31-87), while I used CLIPS (4:1-98), an inherent forward-chainer. The basic difference between forward and

backward-chaining is that backward-chaining starts with the goals, and works back to find applicable facts or sub-goals (3:100-102), while forward-chaining fires any rules if there are applicable facts (3:102-105), and then reaches the goal.

CLIPS allows the easy insertion or deletion of rules. If a new rule is needed, it can be added without any other modification to the code. What must be watched is the order in which the rules are listed, as well as the side-effects of the new rules. With PROLOG, facts and rules can be added quickly, but with greater modification to the rest of the code.

Both forward and backward-chaining need to remember states already tried. Forward-chaining remembers patterns already tried, for a rule, and retains them in memory until there are no other combinations of facts to pattern match, or the rule has been satisfied. Backward-chaining needs to remember all prior goals up until the main goal, as well as the pattern matching for the individual goal.

CLIPS has many more built-in predicates than PROLOG does. Many of the functions that had to be written to do the extraction in PROLOG were included in CLIPS(4:1-98). Another advantage of CLIPS is the ability to compile the CLIPS source programs on any machine that has a C compiler.

One big advantage a backward-chainer (PROLOG) has over a forward-chainer (CLIPS) is the ability to back up and try

different permutations to satisfy a goal. The lack of this ability in a pure forward-chainer is one disadvantage that CLIPS has, and necessitated some extra programming to get the same result.

Another disadvantage of CLIPS is its inability to do list processing and recursion. These processes, used together, would make the extraction problem more tractable. List-processing would allow different permutations of interchangeable inputs to be tried with a single rule, while recursion would allow a rule to handle components that might have any number of interchangeable inputs. The lack of these abilities, as well as the inability in CLIPS to bind an OR clause (3:26,34), necessitates the need for a unique identifier with components that have two or more interchangeable inputs or outputs.

Even though CLIPS has several disadvantages, the advantages of ease of programming, readability of code, and portability of the CLIPS program outweighs these disadvantages.

Problems

A number of problems complicated the extraction of the higher-order logic gates. These problems included running out of memory, slowness of execution, different representations of the same component using the same inputs in different order

and multiple copies of the same circuit.

The memory used by the program is directly related to the number of components in the original file and the number of rules in the program, as well as being exponentially related to the number of pattern-matches needed to extract the higher-order components. The problem of running out of memory was reduced by running the program on a UNIX machine, instead of a DOS machine. Since UNIX does not have a limit of 640k, the program is able to access greater amounts of memory. The Galaxy system, an ELXSI 6400 that was also used, has 64 megabytes of main memory.

Another way to reduce the amount of memory needed is to break the file of rules into a number of smaller files. Each file contains a subset of related rules. This splitting of the rules substantially reduces the memory needed. With all the rules for the first level of extraction in a single file, a DOS based machine with 640k of useable memory could not finish the extraction. Separating those rules into two files allowed the DOS machine to finish the extraction.

The last part of the solution is to run only part of the net-list at one time. As explained earlier, the problem is inherently parallel. Using only part of a net-list at a time, and combining the results, uses much less memory since all the net-list is not in memory at the same time and the amount of patterns that need to be checked is exponentially smaller.

The methods used to take care of the memory problem also take care of the speed problem. Breaking up the rules and the net-list decreased the execution time by a factor of two. The sum of the time needed to do the net-list in parts, and to use only part of the rules is less than the time needed if all the net-list is processed at once with all the rules. This is because the number of rules add to the processing time *exponentially, instead of linearly.*

The problem of keeping only one copy of each component in the fact base at any given time was the most difficult problem to solve at the beginning. There are two ways that a component can be duplicated in the fact base.

The first way is an exact duplicate component, except for the position coordinates. Two transistors can be in a "cif" file with the same node names and different x and y coordinates (there can be more than one transistor with identical gates on a VLSI circuit, replicated to handle speed and power requirements). The only difference between these components is their position information. Only one transistor with each set of node names is needed for the extraction of logic gates. Extra copies of transistors will cause errors in the extraction process. These duplications are handled by the CLIPS routines that remove exact duplicates as well as give each individual transistor a unique identifier.

The second form of duplication occurs if a component has

interchangeable inputs. Two components could have the same inputs, but in a different order. If the inputs are interchangeable, they do not have to be in the same order to make the components identical. A transistor's source and drain are interchangeable, so a transistor of the form (gate, source, drain) is identical to one of the form (gate, drain, source). Transistors are not the only logic components that cause this problem. Any component with interchangeable inputs or outputs, such as NAND gates or NOR gates, has this problem. Back-tracking can take care of this problem in a backward-chainer, but extra work had to be done when using CLIPS.

CLIPS code was written to help alleviate the problem of interchangeable nodes for a transistor. A unique identifier is added to each transistor, checking for duplicates and removing them before any logic extraction is done. This unique ID allows the program to try different input and output combinations, then use the one that matches the pattern. As each transistor is given a unique identifier, the fact base is checked to ensure that no transistor already has those exact node names for inputs and output. If there is no duplicate, the transistor is asserted with a unique identifier. If there is a one, the transistor is retracted. Figure 8 contains example CLIPS code that does the removing of duplicates, as well as the addition of a unique identifier.

```
(defrule add-id-n
  (declare (salience 101))
  ?n <- (n ?gate ?drain ?source ? ? ?x ?y)
  (not (ntrans ? ?gate ?drain ?source ? ?))
  (not (ntrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?n)
  (assert (ntrans =(gensym)
          ?gate ?drain ?source ?x ?y)))

(defrule del-id-n
  (declare (salience 101))
  ?n <- (n ?gate ?drain ?source ? ? ?x ?y)
  (or (ntrans ? ?gate ?drain ?source ? ?)
      (ntrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?n))
```

Figure 8. CLIPS Code to Add ID or Delete Transistor.

This solution will work for components with two or three interchangeable inputs or outputs. For more than that number interchangeable inputs or outputs, the number of combinations becomes too great. The number of permutations needed to be tried for each component increases exponentially with the number of interchangeable inputs or outputs the component has. For two, two combinations needed to be tried, for three, six combinations needed to be tried, and for four, twenty four permutations need to be tried. The number of permutations in the order of the interchangeable inputs or outputs that needs to be checked is **n!**, where **n** is the number of interchangeable inputs or outputs.

For multiple components, each with three or more

interchangeable inputs or outputs, the problem is even worse. As explained earlier, multiple inputs or outputs which are interchangeable mean that more patterns need to be checked. The multiplying factor is $(a!)^b$, where **a** is the number of interchangeable inputs or outputs in a single component, and **b** is the number of components that must be matched that have that number. For components with more than three interchangeable inputs or outputs, using a unique identifier will not be able to replace lists, recursion, and backtracking. The number of different patterns that would have to be checked is prohibitively high. For the scope of this thesis, this limitation did not cause a problem, but for a general purpose extraction and display system, it would.

Results

The extraction program has been tested with a 2400-transistor circuit on the GALAXY computer, an ELXSI 6400 with 10 CPUs rated at 8 MIPS each. It took 120 minutes of cpu time to process the list. It was also tested on a 120-transistor circuit, on a 20 Mhz 386 based computer running DOS, rated at 2.5 mips. The extraction took less than 5 cpu seconds.

At present, the programs can handle transistors, inverters, NAND-gates, NOR-gates, clocked inverters, buffers, multiplexors, exclusive NOR-gates, exclusive OR-gates, and

D flip-flops. As more types of components are needed, they can be easily added.

Components with multiple interchangeable inputs or outputs cause problems in the logic extraction. Adding a unique ID to each component can help alleviate the problem for components with only two or three interchangeable inputs or outputs. For components with more than three, the problem can be resolved using recursion, lists, and backtracking -- which CLIPS does not have in the present version.

III. Graphical Display

Introduction

This chapter discusses the second part of the process of creating a graphical display. It covers how the graphical representation of the extracted components is achieved. The sections of the chapter are the tasks to be done, procedures followed to achieve the graphical display, and problems encountered. The part on procedures to be followed is broken into smaller sections described in the tasks section. The final section of this chapter is a summary of results. A user's manual on how to use the graphical display is in appendix D.

Tasks

The first part of the problem of creating a graphical display, representing a circuit symbolically and extracting higher-order logic gates from a net-list, was discussed in the previous chapter. The solution to the remaining part of the problem, how to graphically represent the extracted components, is presented in this chapter.

Graphically representing a circuit is the displaying of the components of a net-list on a graphics device, i.e., showing the standard symbol for each extracted component on a computer screen at the same relative position that it

occupies in the circuit. The problem of displaying the circuit decomposes into five smaller tasks.

The first task is to find the correct relative position of the extracted components. That task is taken care of during the extraction routines. The second task is to find what the maximum and minimum x and y coordinates of the chip are, so that the graphics display can be correctly scaled. The third task is to correctly parse the file containing the extracted components, and to pass the information to the graphics routines. The fourth task is to initialize a graphics surface, allowing input from the user. The fifth and last task is to draw the components on the screen, labeling them with the names of their connected nodes.

Finding the Relative Location of Components

The first task, finding the correct relative location for the extracted components, is handled by the extraction routines. The new x and y coordinates for the extracted component are the averages of the x and y coordinates of the lower-order components composing the higher-order gate. This method was chosen because it was the simplest way to assign coordinates to a newly extracted component. Figure 9 shows an extract of the CLIPS code which extracts an inverter and gives it a new location. The function "bind" binds the value of the solved expression to a variable (3:34). If there were

n lower-order components, then the new x and y locations would be the sum of the x and y coordinates of the components divided by n.

```
(defrule inverter
 (or (ptrans ?id1 ?gate vdd ?a ?x1 ?y1)
     (ptrans ?id1 ?gate ?a vdd ?x1 ?y1))
 (or (ntrans ?id2 ?gate gnd ?a ?x2 ?y2)
     (ntrans ?id2 ?gate ?a gnd ?x2 ?y2))
 ?p <- (ptrans ?id1 $?)
 ?n <- (ntrans ?id2 $?)
=>
 (bind ?xa (/ (+ ?x1 ?x2) 2))
 (bind ?ya (/ (+ ?y1 ?y2) 2))
 (retract ?p ?n)
 (fprintout component "(inv "?gate" "?a" "?xa"
                                    "?ya")"crlf))
```

Figure 9. CLIPS Code to Extract Inverter and Find New x and y Coordinates

It is possible that two extracted components might be given the same, or overlapping locations. This problem is discussed in the problem section.

Finding the Extrema of the Component Coordinates

The next task is finding the maximum and minimum x and y coordinates of the components. These extrema are needed to determine the size and shape of the SunCore window when it is initialized. This task is handled by a CLIPS routine "findext.clp" that is run after the extraction is complete.

The code for "findext.clp" is in appendix A.

The algorithm implemented in "findext.clp" finds an x coordinate, then finds an x coordinate larger than the one before, checking the whole fact base until the largest x coordinate is found. The minimum x coordinate is found in the same manner, as are the largest and smallest y coordinate. After the largest and smallest x and y coordinates are found, they are written to the file "scaled.clp", in the form

(range xmin xmax ymin ymax)

where xmin is the minimum x, xmax is the maximum x, ymin is the minimum y, and ymax is the maximum y. The components are then written to the file one at a time, converting the x and y coordinates to integers as they are written. Figure 10 is an example of this type of file, using the clock generator circuit described in the next section. The first line is the maximum and minimum x and y coordinates for this circuit, and the x and y coordinates have all been changed to integers.

Parsing the CLIPs file

The file that is produced by "find.ext" must be parsed to pass the values to the graphics routines. This is done by reading each line into a buffer, stripping off the parentheses, then using the SSCANF() (5:246) function of C to separate the different parts of the line and bind the values to variables. Since the graphics routines are written in C,

```
(range -20587 74850 -10050 6112)
(inv 444 450 14925 -1050)
(inv 449 443 5175 -900)
(inv 450 424 17025 -675)
(nor2 gen45 329 424 447 20475 -675)
(nor2 gen44 447 453 329 24262 -525)
(inv 424 453 27450 0)
(inv IZ_go 498 1500 -225)
(nand2 gen43 498 450 449 3150 -225)
(inv 53 OZ_pq2 74850 -4425)
(inv 329 277 18600 -8925)
(buffer 277 53 32850 -10050)
(buffer 443 444 8925 -1050)
(buffer 447 533 300 6112)
(buffer 533 OZ_pq1 -20587 412)
```

Figure 10. Circuit File with Extrema

a function call with parameter passing is needed to pass the values to a graphics routine. The code for "drawit.c" which does the stripping and parsing, as well as the rest of the graphical display, is in appendix B.

Initializing a Viewing Surface

Finding the correct graphics routine to use on the Suns was the hardest part of creating a graphical display. Each routine has its advantages and disadvantages. Overall, SunCore (6:1-223) is the graphics package that is easiest to use. SunCore commands are simple, have a direct mapping of values to x and y moves, and was the easiest to learn in a short time span.

The limitations of SunCore are that it allows drawing

using only lines, polylines, shading, and text (6:53-72). Polylines are connected continuous lines. It does not have built-in primitives for curves or circles. Despite this lack, its graphics commands are the simplest, once a viewing surface is initialized. A viewing surface must be initialized since SunView allows multiple windows. The graphics routines need to be told which surface or window it will be drawing in.

Another problem is that without massive programming, SunCore can only draw in the calling window. This problem was solved by creating a batch file which creates a new window of size 900 by 900 in the upper left corner of the Sun's screen, then calls the graphics routine from this window. That batch file, "drawgate", is included in appendix B.

Like the graphics program's need to be told where to draw, the mouse needs to be initialized to the display window and told where to look for button inputs, and where to show its icon. Mouse support is needed to allow easy exit from the graphics display window back to the data input window.

Drawing the Component Symbols

The functions that actually draw the symbols are called from within "drawit.c". When the gate type has been determined by "drawit.c", a retained segment is opened and the appropriate function is called with the parameters for node names and x and y coordinates passed to it.

Each type of component that can be created by the extraction process has a corresponding function which will draw it on the screen. Depending on the level of extraction, the symbols are drawn on the screen in different sizes. Supposing the smallest components are unit size, components at the next level of extraction are roughly two units by two units, and the next level roughly four units by four units. Each subsequent level of extraction has its symbols roughly doubled in size. Symbol size, as well as size of the text of the node names, changes with the level of extraction of the component. Figure 11 is a list of the symbols so far implemented, along with their level of extraction, 0 being the lowest level. Figure 12 is a screen from the graphics display showing all the symbols implemented so far, and their relative sizes.

ntrans	0
ptrans	0
inv	1
tgate	1
nand2	1
nor2	1
clk_inv	1
buffer	2
mux	2
xnor	2
xor	2
dff	2

Figure 11. Size Level of Implemented Symbols

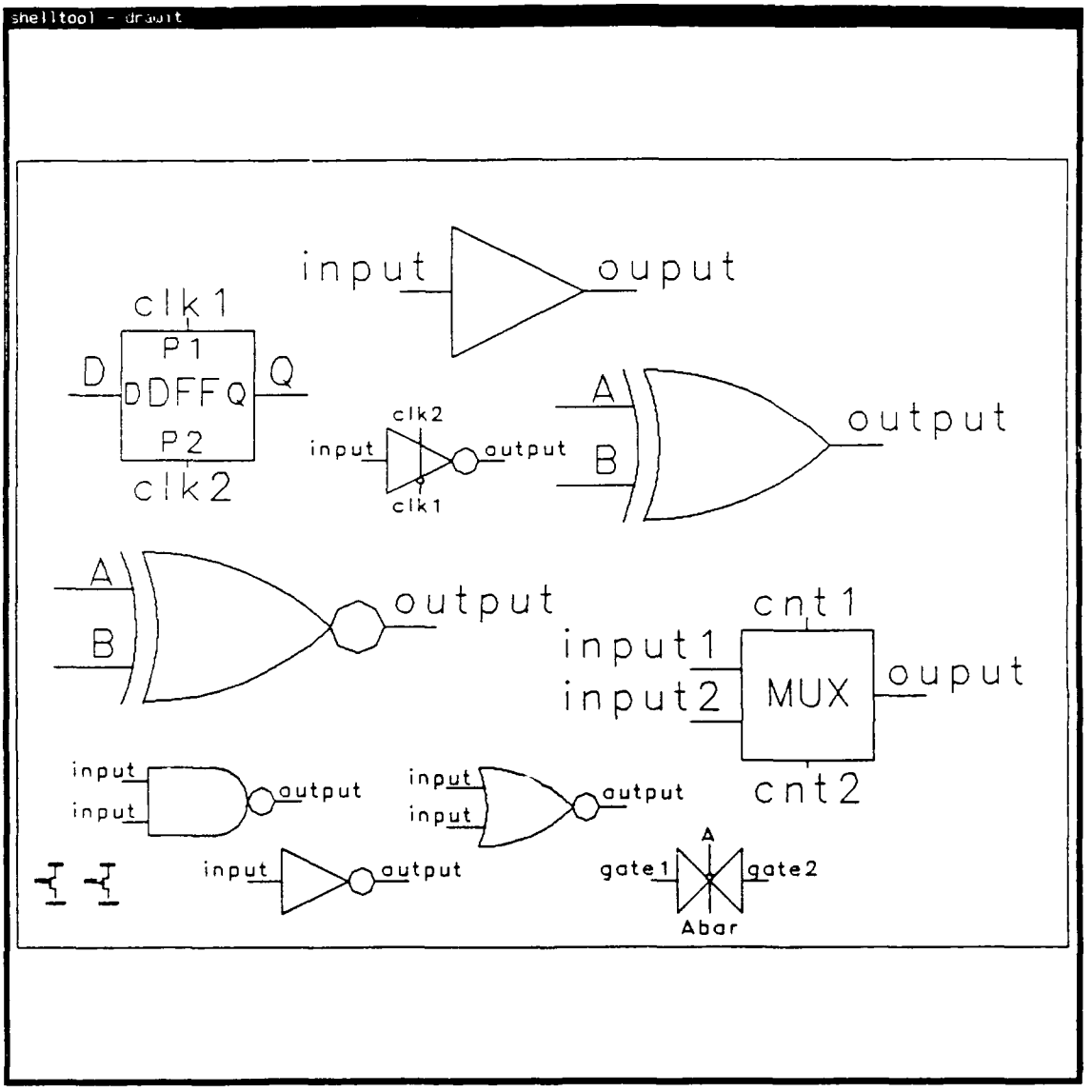


Figure 12. Symbols Implemented by the Display

The basic structure of the functions to draw all the symbols is identical. The code starts with a pair of static variable structures which hold a list of x moves and y moves. The function moves to the beginning x and y coordinate, with an offset determined by the level of extraction of the component being drawn. A polyline is then drawn, using the data in the pair of structures. Finally, the node names are added after the character size is set. Additional polylines are added if the symbol is cannot be drawn with a single continuous line. Figure 13 is the function to draw a buffer. It is an example of a function to draw a basic symbol. The functions to draw the rest of the symbols are located in appendix B.

A test driver program , "testgate.c", was written to assist in designing all the symbols, by allowing the display of each symbol in the same location on the graphics display by inputting the symbol name. During the design of the symbols, each one had to be drawn repeatedly. This program, "testgate.c", allowed display of any symbol, without using a circuit file. The program "testgate.c", and the associated batch file "drawtest", are located in appendix C.

Problems

As mentioned earlier, two of the problems encountered were the inability to draw in a separate window, and the lack

```
#include <usercore.h>
#include <string.h>

static float bufferx[] = {1000.0, 400.0, -400.0,
                          -1000.0, 0.0, -400.0,
                           400.0, 0.0};

static float buffery[] = {500.0, 0.0, 0.0, 500.0,
                           -500, 0.0, 0.0, -500.0};

buffer(x0, y0, gate1, gate2)
float x0, y0;
char *gate1, *gate2;
{
    move_abs_2(x0+2000.0, y0+500.0);
    polyline_rel_2(bufferx, buffery, 8);
    set_charspace(50.0);
    set_charsize(160.0, 160.0);
    move_rel_2(-210.0*(strlen(gate1)+0.5), 670.0);
    text(gate1);
    move_abs_2(x0+3100.0, y0+1170.0);
    text(gate2);
}
```

Figure 13. Function to Draw a Buffer

of built-in primitives to draw curves. Other problems that occurred were the inability to see program prompts from the graphics screen, and to reply to them. The most serious problem, for which no solution has been implemented, is the possibility that two or more extracted components have the same x and y coordinates. A similar problem is how to scale the symbols correctly so that they will not overlap if the x and y coordinates of two symbols x are too close to each other. All the problems, except the last two, were solved in a workable manner.

The inability to draw in a separate window, and the problem of getting and giving program prompts, are related. The difficulty is that the program cannot write to the screen as standard output while the screen is initialized in graphics mode. No other window can be written to, either. The problem was solved by switching the window between graphics and text mode. After user choices are made, the program switches to graphics mode, and displays the graphical display until the right mouse button is pushed while the pointer is in the graphics window. Once the mouse button is pushed, the window switches back to text mode to get more user inputs.

The problem of the lack of primitives to draw curves was solved by writing a C program "circlept.c". This program asks for the number of points, and the radius, then gives the x moves and y moves to draw a circle of the given radius, composed of that number of points. The points start and end on the extreme left side of the circle, since the major use of the program was to draw the inverter bubbles on the right side of some of the gates. The code for "circlept.c" is in appendix B.

The problem of two different extracted components having the same coordinates is caused by two different factors. The first factor is the simple method in which the coordinates are averaged. Since the new coordinates are just the averages of the old coordinates, duplication of coordinates can occur, if

the averages of two different sets of component come out to be the same. The second factor is the choice of transistors kept out of duplicate sets. The first transistor encountered is the one kept, although another transistor might have a better location. Since all locations derive from the locations of the transistors, this is part of the cause of the problem. The way to solve this problem would be to have the rule-base system check the coordinates of all other components before assigning new coordinates to a newly extracted component. This solution was not implemented due to time constraints placed on this research, and the increase in processing time it would cause.

The problem of overlapping symbols is caused when two components are too close to each other. This problem is caused by the same factors that caused the previous problem. A solution to this problem, which was also not implemented due time constraints placed on this research, is to allow the symbols' sizes to be scaled. This solution would not stop all overlaps, but would stop most.

The problem of overlapping symbols increases with the decrease in the size of the technology used to design the circuit, as well as increasing with the number of transistors used in the circuit. These factors affect the overlaying of two components exactly the same way.

Results

The graphical display of symbols of components was achieved by using the built-in SunCore graphics routines of the Sun workstation. Processing time to display the graphics is negligible, taking under a second. The majority of the processing time is used to open the SunView window in which the graphics are drawn. On a Sun 3/50 monochrome machine, the creation of the window takes 5 seconds. It takes less time, 3 seconds, to set up the window on the color Sun 3/60 system. This is due to the fact that the Sun 3/60, with 8 megabytes of RAM, can continue more processes without swapping memory, than the Sun 3/50, which has 4 megabytes of RAM. The time taken to read the CLIPS data file is also negligible for the small circuit of 120 transistors tested.

IV. Clock Generator Circuit Example

Introduction

A single VLSI design was used to determine the performance of the logic extraction and the graphical display. Though a minimal test, this extraction and display provides a proof of concept for (a) logic extraction using CLIPS, and (b) graphical display on a Sun 3 workstation. It also demonstrated the need for more capabilities than were designed into the system. Conclusions and recommendations are discussed in the next chapter.

The circuit that is used as an example is the clock generator used by CPT Dukes as an example in his thesis (2:76-78). One reason this circuit was chosen was because it consists entirely of inverters, NOR gates, and a NAND gate; all implemented using Static CMOS design. Therefore no transistors were left after extraction. The count of only 116 transistors, and only 42 unique transistors, in the circuit allowed design and execution of extraction program on a DOS-based machine with only 640k of memory available to the program. The low component count of the extracted circuit made testing the graphical interface simpler. Another reason was that the higher-order logic of the circuit was already known, and could be used to judge the "correctness" of the extraction. Figure 14 is a circuit diagram of the clock

generator, as designed. The example is described in two sections: The first concerns the extraction; the second, the graphical display. A summary of the example is included at the end of the chapter.

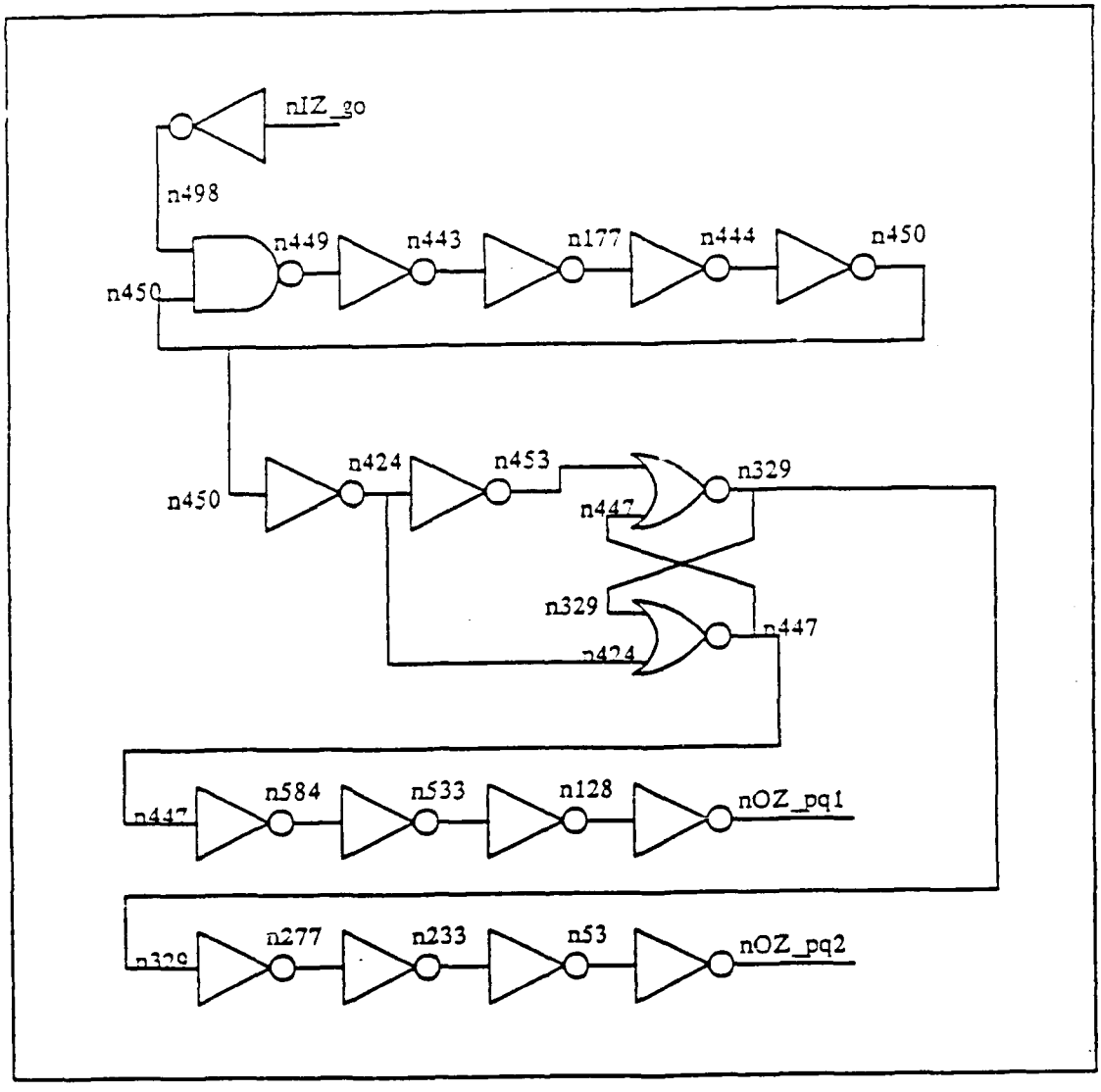


Figure 14. Circuit Diagram of Clock Generator (Figure from 2:78)

Extraction

The first step of the extraction was the conversion of the form used by the "cif" file for the clock generator circuit into the form that CLIPS can read. This conversion took less than one second on a 20Mhz 386 DOS machine using the program "sim2clip". Both the original "sim" file and the CLIPS file contain 116 transistors. Duplicate transistors found in these files are removed during the extraction process. Due to the length of the files, the "sim" file of the clock generator circuit and the CLIPS file "good.clp" are included in appendix B, rather than being presented in this chapter.

The extraction process, using the CLIPS files tr1.clp, tr2.clp and tr3.clp, took less than 5 cpu seconds on Galaxy, an ELXSI 6400; less than 5 cpu seconds on a Sun 3/60; and less than 5 seconds on a 20Mhz 386 DOS machine. Finding the extreme x and y coordinates took less than a second on all the machines. Figure 15 is the extracted file of components for the clock generator circuit.

Comparing the extracted components in figure 15 to the circuit diagram in figure 14 shows only a few differences. These differences can all be attributed to the combination of pairs of inverters into buffers. Logically the components in the circuit diagram in figure 14, and the components listed in figure 15 are the same. The extraction process was done

```
(range -20587 74850 -10050 6112)
(inv 444 450 14925 -1050)
(inv 449 443 5175 -900)
(inv 450 424 17025 -675)
(nor2 gen45 329 424 447 20475 -675)
(nor2 gen44 447 453 329 24262 -525)
(inv 424 453 27450 0)
(inv IZ_go 498 1500 -225)
(nand2 gen43 498 450 449 3150 -225)
(inv 53 OZ_pq2 74850 -4425)
(inv 329 277 18600 -8925)
(buffer 277 53 32850 -10050)
(buffer 443 444 8925 -1050)
(buffer 447 533 300 6112)
(buffer 533 OZ_pq1 -20587 412)
```

Figure 15. Extracted Clock Generator Circuit

on the ELXSI, a Sun 3/60, and a 80386 based DOS machine, with identical results.

Display

The example display of the whole circuit, discussed later in this chapter, was done on a monochrome Sun 3/50 workstation with 4 megabytes of RAM. The actual creation of the display took less than one second. It takes between 4 and 10 seconds to create the shelltool window in the Sunview windowing system, however, depending on how many other windows are open in the environment. This time lag is only for the first display in a session, not for subsequent displays in the same session.

This shelltool window is created by running batch file

"drawgate", which sets up a shelltool window of known size and starts the actual drawing program "drawit.c" in the window. The known size is needed to get correct aspect ratios for the displayed symbols. Figure 16 is the batch program "drawgate". The code for "drawit.c", as well as the functions to draw all the components, are listed in appendix B.

```
#!/bin/csh -f
shelltool -Wp 0 0 -Ws 900 900 "drawit"
```

Figure 16. Batch File to Open Window

After the batch file was run, the name of the file containing the circuit was asked for, as well as the coordinates of the window of the part of the circuit to view.

Two different displays of the circuit are included to show the resolution of the display depending on the portion of the circuit displayed. The first display, shown in figure 17, is of the whole circuit. The values entered as the maximum and minimum values for x and y were the extreme values found by the extraction routines. This display is shown to illustrate the limitations of viewing the whole circuit at once.

From figure 17, it is obvious that the details of the components cannot be discerned when the whole figure is displayed. A look at the complete circuit does help, however,

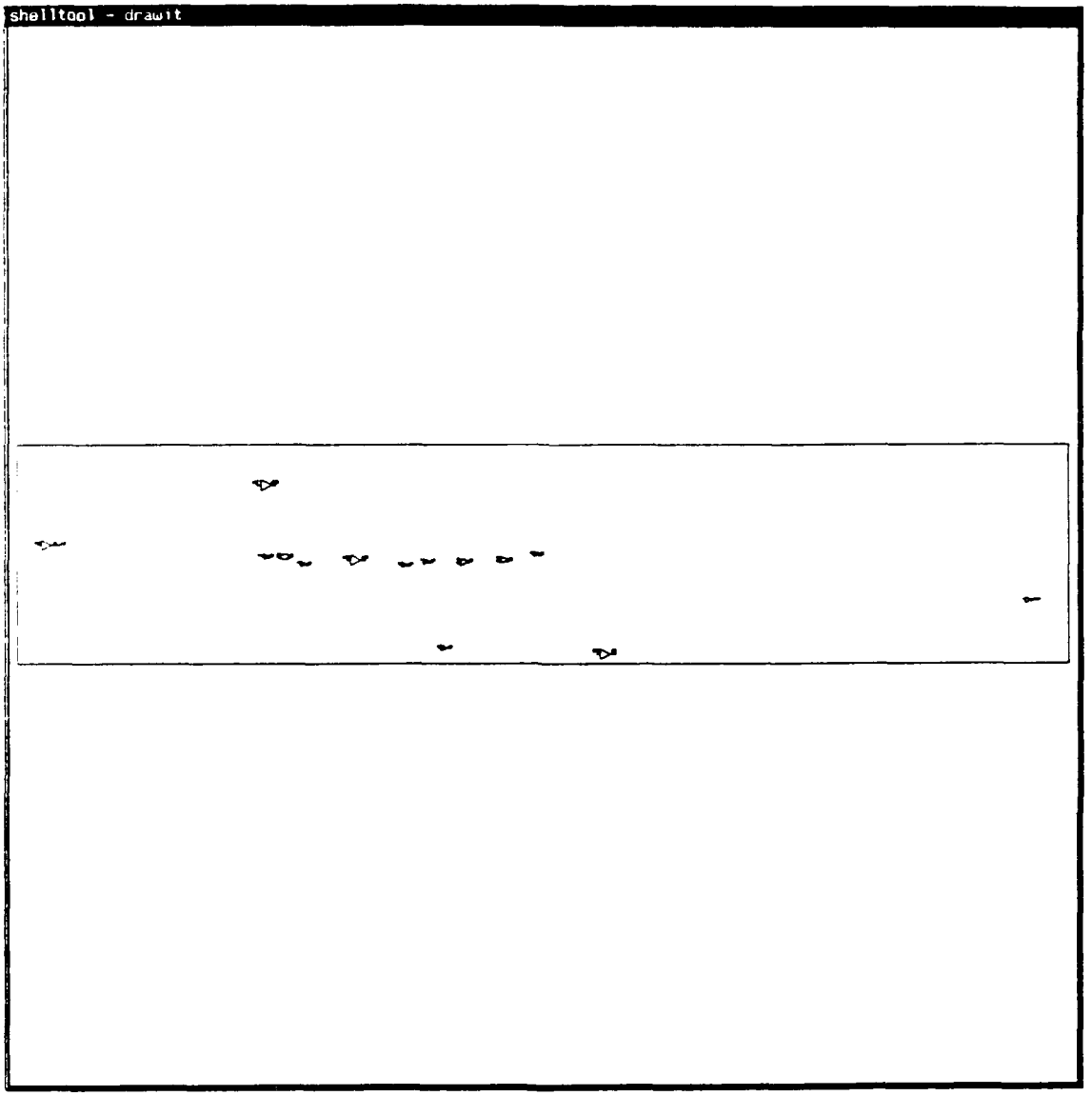


Figure 17. Display of Whole Circuit

to decide where to zoom for further views. It also shows the number of components that are in the extracted file, though without enough resolution to discern their node names, or even what some of the components are.

The second display, shown in figure 18, is a closeup of part of the circuit. This display was chosen to illustrate the legibility and usefulness gained by viewing part of the circuit. Figure 18 is a closer look at the part of the circuit within the x range of 16000 to 30000, and the y range from -10000 to 8000. This view is of a large enough portion of the circuit to show connections between different components, yet small enough so that the components and their interconnections are clearly identifiable. This display is of the type which would be the most useful for an engineer doing reverse engineering, or original design. Using the coordinates from MAGIC for part of a circuit, an engineer could determine what that part of the circuit is. This is the real power of the graphical interface.

Figure 19 is the session that produced both aforementioned displays.

Summary

The task of extracting the higher-order logic components of the clock generator circuit was accurately completed by the CLIPS extraction routines. The extracted components were

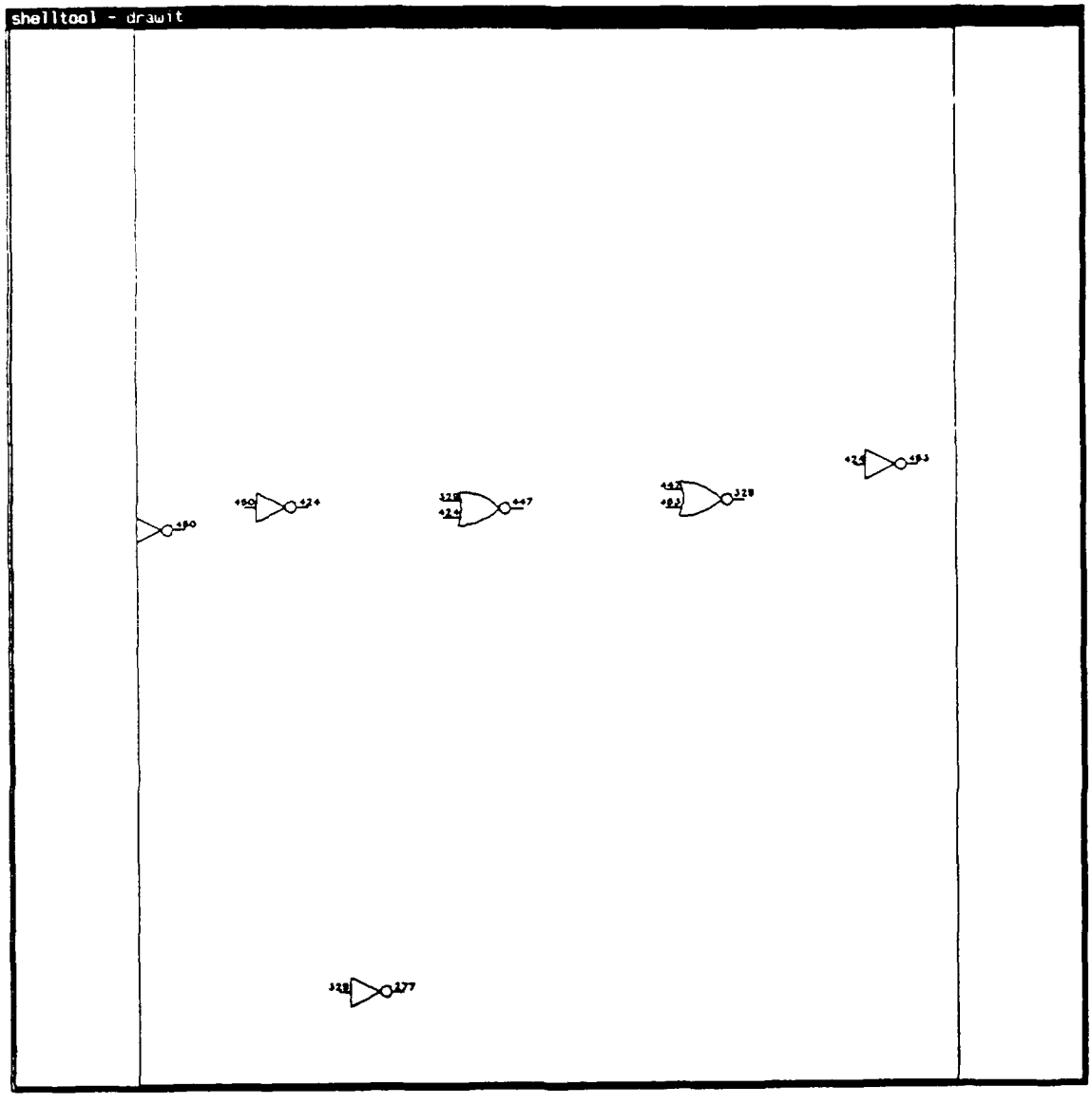


Figure 18. Close-Up Display of Circuit

logically identical to components shown in figure 14. The graphical display symbolically showed the connections between the different components, or the relative location of components, depending on the view displayed. The extraction and display of the clock generator circuit demonstrated that the extraction routines work, and that the graphical display program is capable of reading a CLIPS file and displaying it on a Sun 3/50 or Sun 3/60 workstation.

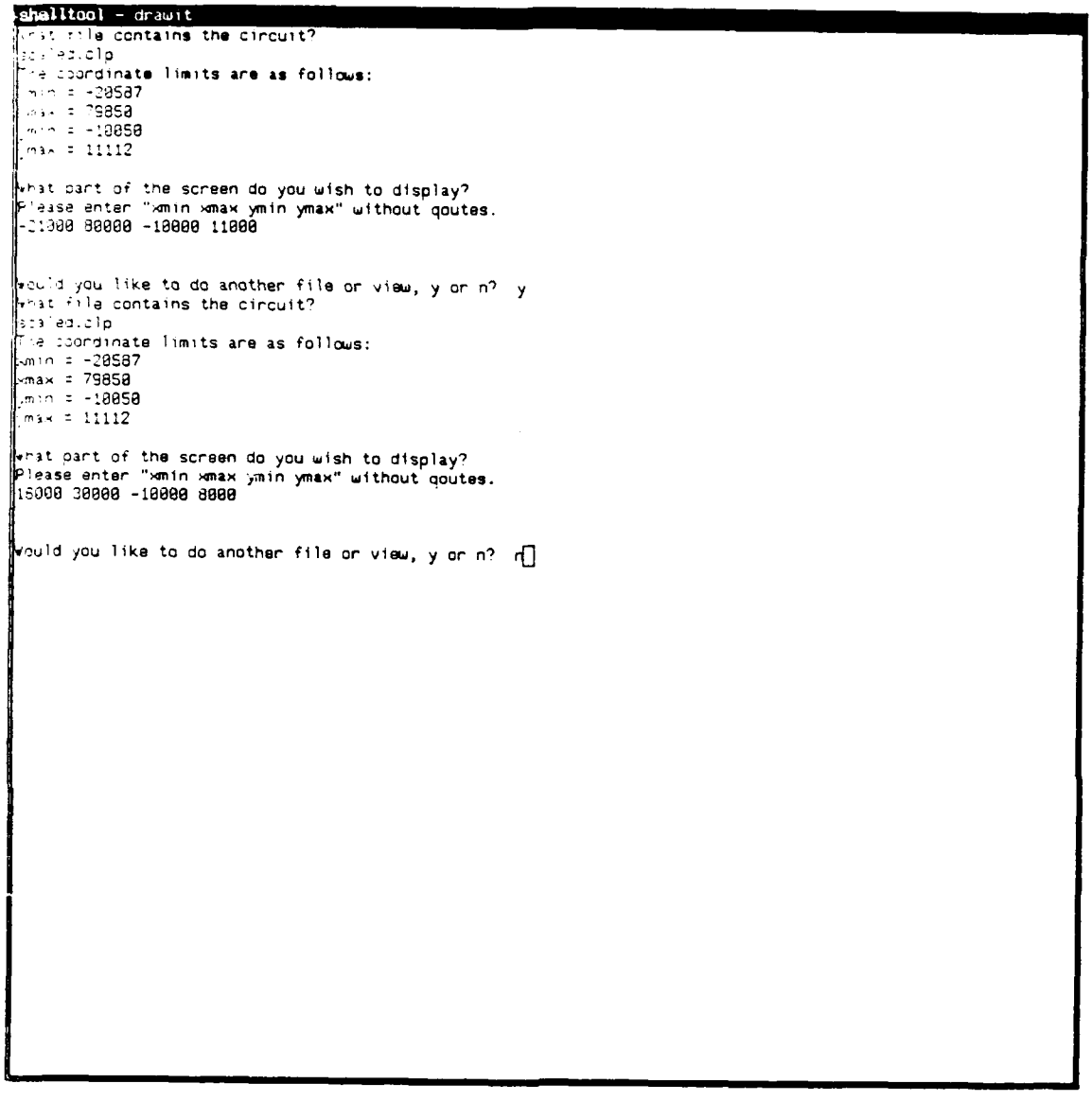


Figure 19. Display Input Session

V. Conclusions and Recommendations

Introduction

In this chapter, conclusions and recommendations resulting from this research are presented. Suggested improvements of the extraction system and graphical display are discussed and topics for follow-on theses are presented. A summation of the project is included at the end of this chapter.

Conclusions

CLIPS, a forward chainer, has advantages and disadvantages over PROLOG, a backward-chainer. The advantages include quick implementation of understandable rules and easily divisible code. Another advantage is the ability of CLIPs to run identical code on any system that has a C compiler. Time of execution to extract a circuit seems to be comparable for both CLIPS and PROLOG. Both took under 10 seconds for the clock generator circuit (2:77). Disadvantages include the need to work around the lack of back-tracking, recursion and list-processing; abilities which are inherent in PROLOG. The addition of these abilities in a later version of CLIPS would make it a more suitable choice for a project of this type. The lack of these abilities hinders the usefulness of CLIPS when dealing with many levels of extracted

components. Overall, the use of CLIPS as the language for the circuit extraction portion of this thesis showed that a forward-chainer can do a job comparable to PROLOG with the few aforementioned exceptions, producing a workable circuit extraction system.

The use of SunCore graphics as the graphics package to implement the graphical display portion allowed quick programming of the interface. With the ease of programming come certain limitations. The inability to produce graphics in any window other than the current window in SunView hindered the development of the interface. More time to learn about SunCore graphics might have revealed a way around this problem.

The graphical display program, as designed, can display any number of circuit components if they are in a file of the proper form. The graphical display program will scale the size of the components, dependent upon the portion of the circuit being drawn. As less of the circuit is displayed, the individual components are drawn larger, and as more of the circuit is displayed, the components are drawn smaller. This feature allows both the zooming and panning of the extracted circuit by the choices of the minimum and maximum x and y coordinates displayed. It also allows precise control over what portion of the circuit is displayed.

This method of display has some inherent problems. The

program will blindly place a component wherever the file says there is one, overlapping or covering previously drawn components. This problem is minimized by the scaling of the size of the components. Unfortunately, the extraction routine can place two or more components into too-close proximity. Scaling cannot remove all overlaps.

Overall, the graphical display does what it is designed to do. It displays higher-order logic components that have been extracted from a net-list, on a Sun 3 workstation, in the relative positions they occupy on the chip. It allows zooming and panning, as well as sequential view of different files. As implemented, the circuit-extraction system and graphical display can help an engineer design a circuit, or help reverse engineer an unknown circuit.

Recommendations

The recommendations that follow from this thesis can be divided into three different portions: improvement of the extraction routines, improvements to the graphical display, and extensions to the concept of a graphical interface. All recommendations would fill deficiencies that have been discovered during the course of implementation of the extraction system and graphical display.

During the course of this thesis, the ability for CLIPS to perform logic extraction has been confirmed. The ability

of CLIPS to run identical code on any system that has a C compiler is a big advantage over other AI languages, such as PROLOG or LISP, which have limited numbers of platforms, or lack of compatibility due to different dialects of the languages running on different platforms. For these reasons, the use of CLIPS for logic extraction should be explored further.

Additional logic components should be added to the system, as well as new levels of extraction. Other changes to the system should also be made to support the other recommendations to be made.

The graphical display, as mentioned earlier, does what it was designed to do, though in a less user-friendly manner than desired. Most of the recommendations to be made concerning the display revolve around ease of use. These recommendations fall into the categories of improving the mouse support, improving the visual display,and improving the use of the features of the SunView environment.

Greater use of the mouse would enhance the friendliness of the system. The ability to use the mouse to pick a window to zoom into would make the system easier to manage. Another use of the mouse would be to pan the portion of the circuit viewed, to create a sliding window effect. These added features would make the system more of a usable CAD tool.

The visual display could be improved in several ways.

Coordinate axes along the sides of the display would help the precise zooming of view, and location of components. The addition of connecting lines between connected nodes on components, instead of just node name labels, would also make the display simpler to comprehend. Another recommendation is to make the symbols all scaleable in size, allowing larger symbols for portions of circuits where individual components are not as near to one another, and smaller symbols where the components are nearer to one another.

Several features incorporating greater use of the SunView environment would improve the graphical display. These include the use of separate command and graphics windows, pop-up or pull-down menu support, and multiple views of the same circuit. The use of SunCore innibits the development of some of these features. To fully utilize the features mentioned, the graphical display should be rewritten to utilize either Sun CGI or Pixwinn graphics.

During the course of the thesis, the limitations of just showing symbols representing extracted components in their relative positions were discovered. Due to sizing problems, the circuit as a whole cannot be seen without making the individual symbols too small. An extension to the concept of a graphical display, which would entail considerable AI programming as well as graphics programming, would be the creation of a circuit diagram from the extracted components.

This display would show all the components extracted in the manner of the circuit diagram in chapter IV, figure 18. An extensions of this magnitude would require four parts. The first part would be assigning new locations to all the components, such that none overlap, and components are near the components to which they are connected. The second part would be rotating the components to ease the problem of routing lines between different components. The third part would be routing the lines. The last part would be creating a program that can display the diagram, zoom in and out, and pan easily over the whole circuit. These abilities would be needed for circuits that have too many components for the individual components to be recognizable at full view. A program of this type could be adapted from the graphical interface created during this thesis.

A thesis implementing the foregoing recommendations would create a greatly needed CAD tool. This tool, in conjunction with the extraction system and graphical display described in this thesis, would help close the CAD cycle, as well as easing the task of reverse-engineering circuits for which no circuit descriptions exist.

Summary

The conclusions presented in this chapter illustrate the value of the research to the engineering community. The

extraction system and graphical display can be used to help design a new circuit, or reverse-engineer a circuit whose function is not totally known.

Appendix A: Extraction Code

```
     SIM2CLP.C, conversion program from "sim" to CLIPS.
/*********************************************************
*
*    Date:  8 November 1989
*    Version: 1.0
*
*    Title: sim2clip Translation Routine
*    Filename: sim2clip.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3, MS-DOS V3.3
*    Language:  C
*    Description:
*         This routine takes a transistor net-list
*         from an ESIM file produced by MEXTRA, and
*         generates a CLIPS formatted description for
*         the same file. All Fields are used.
*
*    Passed Variables: None
*    Returns: None
*    Files Read: new.sim
*    Files Written: good.clp
*    Documentation:
*         This program is a modification of sim2pro,
*         written by Capt Mike Dukes for his thesis
*         (2:127).
*    Special Instructions : None
*********************************************************/

#include <stdio.h>
#define max_buf 128
char buffer[max_buf];          /* Holds the origional line.*/
char tempbuf[max_buf];         /* Holds the converted line.*/

/* Variables */

int iteration, count,count2;
FILE *fd,*od;

main()
{
     /* Opens input and output files */

fd=fopen("new.sim","r");
od=fopen("good.clp","w");
```

```
     /* Clearing the temporary buffer */

for(count=0;count<max_buf;count++)
     {
     tempbuf[count]=0;
     }

     /* Get first line, which is thrown away. */

fgets(buffer,max_buf,fd);
while(fgets(buffer,max_buf,fd) != NULL)

     /* get next line, if it exists */

     {

     /* Check for N type transistor */

     if(buffer[0]=='e')
          {

     /* Set variables, then translate the first three
           characters in the buffer. */

          count=3;
          count2=2;
          iteration=0;
     tempbuf[0]='(';
     tempbuf[1]='n';
          tempbuf[2]=' ';

     /* Check to insure there are characters left, and
           that only 7 fields are translated. */

          while((buffer[count2]!=0)&(iteration!=7))
               {

     /* Check for a Vdd, and translate it to a vdd. */

               if ((buffer[count2]=='V')&(buffer[count2+1]=='d')&
                  (buffer[count2+2]=='d'))
                    {
                    tempbuf[count++]='v';
                    tempbuf[count++]='d';
                    tempbuf[count]='d';
                    count2=count2+2;
                    }

     /* Check for a GND, and translate it to a gnd. */
```

```
        else if ((buffer[count2]=='G')&
                 (buffer[count2+1]=='N')&
                 (buffer[count2+2]=='D'))
            {
            tempbuf[count++]='g';
            tempbuf[count++]='n';
            tempbuf[count]='d';
            count2=count2+2;
            }

/* Check for a blank, and write it. */

        else if(buffer[count2]==' ')
            {
            tempbuf[count]=' ';
            iteration++;
            }

/* Check for a # and delete it. */

        else if(buffer[count2]=='#')
            {
            --count;
            }

/* Write anything else as is. */

        else
            {
            tempbuf[count]=buffer[count2];
            }
        count++;
        count2++;
        }

/* Write the closing ")" and crlf. */

        count=count-1;
        tempbuf[count++]=')';
        tempbuf[count++]=10;
        tempbuf[count]=0;

/* Write the tembuf back to the original buffer, then
      write it to the output file. */

   for(count=0;count<max_buf;count++)
        {
        buffer[count]=tempbuf[count];
        }
   fprintf(od,"%s",buffer);
```

```
    for(count=0;count<max_buf;count++)
        {
        tempbuf[count]=0;
        }
    }

 /* Check for P type transistor */

else if(buffer[0]=='p')
    {

 /* Set variables, then translate the first three
       characters in the buffer. */

    count=3;
    count2=2;
    iteration=0;
    tempbuf[0]='(';
    tempbuf[1]='p';
    tempbuf[2]=' ';

 /* Check to insure there are characters left, and
       that only 7 fields are translated. */

    while((buffer[count2]!=0)&(iteration!=7))
        {

 /* Check for a Vdd, and translate it to a vdd. */

         if((buffer[count2]=='V')&(buffer[count2+1]=='d')&
            (buffer[count2+2]=='d'))
            {
            tempbuf[count++]='v';
            tempbuf[count++]='d';
            tempbuf[count]='d';
            count2=count2+2;
            }

 /* Check for a GND, and translate it to a gnd. */

        else if ((buffer[count2]=='G')&
                 (buffer[count2+1]=='N')&
                 (buffer[count2+2]=='D'))
            {
            tempbuf[count++]='g';
            tempbuf[count++]='n';
            tempbuf[count]='d';
            count2=count2+2;
            }
```

```
        /* Check for a blank, and write it. */

                else if(buffer[count2]==' ')
                    {
                    tempbuf[count]=' ';
                    iteration++;
                    }

        /* Check for a # and delete it. */

                else if(buffer[count2]=='#')
                    {
                    --count;
                    }

        /* Write anything else as is. */

                else
                    {
                    tempbuf[count]=buffer[count2];
                    }
                count++;
                count2++;
                }

        /* Write the closing ")" and crlf. */

                count=count-1;
             tempbuf[count++]=')';
                tempbuf[count++]=10;
                tempbuf[count]=0;

        /* Write the tembuf back to the original buffer, then
              write it to the output file. */

           for(count=0;count<max_buf;count++)
                {
                buffer[count]=tempbuf[count];
                }
           fprintf(od,"%s",buffer);
           for(count=0;count<max_buf;count++)
                {
                tempbuf[count]=0;
                }
           }
        }
         /* Close both files */
    fclose(fd);
    fclose(od);
    }
```

TRN.BTT, batch file which runs extraction process.

```
(load "tr1.clp")
(reset)
(run)
(clear)
(load "tr2.clp")
(reset)
(run)
(clear)
(load "tr3.clp")
(reset)
(run)
(exit)
```

FINDEXT.BTT, batch file which runs findext.clp.

```
(load "findext.clp")
(reset)
(load-facts "outcomp2.clp")
(load-facts "comprem2.clp")
(run)
(exit)
```

TR1.CLP, first CLIPS file for first level of extraction.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TR1.CLP  written by Capt Stuart Yarost as part of the
;; requirements for MS in Computer Engineering.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This file adds a unique identifier to every transistor in
;; good.clp, filters out unnecessary information, then builds
;; up the following components from the n and p type
;; transistors:
;;   inverters
;;   tgate
;;   nand (2 input)
;;   nor  (2 input)
;; As it writes these components to the file outcomp1.clp, it
;; retracts its  component transistors from the database.
;; When it is complete, it writes the remaining facts to the
;; file compremi.clp.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; load-facts loads the facts from good.clp.

(defrule load-facts
  ?i <- (initial-fact)
=>
  (load-facts "good.clp")
  (retract ?i)
  (assert (open-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; open-file opens outcomp1.clp as a write only file.

(defrule open-file
  (declare (salience 101))
  ?i <- (open-file)
=>
  (open "outcomp1.clp" component "w")
  (retract ?i)
  (assert (close-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; add-id-n adds a unique identifier to each n type
;; transistor, if a similiar n type transistor has not already
;; been asserted, with the same gate, and same source and
;; drain.

(defrule add-id-n
  (declare (salience 101))
  ?n <- (n ?gate ?drain ?source ? ? ?x ?y)
  (not (ntrans ? ?gate ?drain ?source ? ?))
```

```
  (not (ntrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?n)
  (assert (ntrans =(gensym) ?gate ?drain ?source ?x ?y)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; del-id-n deletes a n type transistor if it has already been
;; asserted, with the same gate, source and drain.

(defrule del-id-n
  (declare (salience 101))
  ?n <- (n ?gate ?drain ?source ? ? ?x ?y)
  (or (ntrans ? ?gate ?drain ?source ? ?)
      (ntrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?n))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; add-id-p adds a unique identifier to each p type
;; transistor, if a similiar p type transistor has not already
;; been asserted, with the same gate, and same source and
;; drain.

(defrule add-id-p
  (declare (salience 101))
  ?p <- (p ?gate ?drain ?source ? ? ?x ?y)
  (not (ptrans ? ?gate ?drain ?source ? ?))
  (not (ptrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?p)
  (assert (ptrans =(gensym) ?gate ?drain ?source ?x ?y)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; del-id-n deletes a n type transistor if it has already been
;; asserted, with the same gate, source and drain.

(defrule del-id-p
  (declare (salience 101))
  ?p <- (p ?gate ?drain ?source ? ? ?x ?y)
  (or (ptrans ? ?gate ?drain ?source ? ?)
      (ptrans ? ?gate ?source ?drain ? ?))
=>
  (retract ?p))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; inverter writes an inverter, averaging the positions, and
;; retracts the component transistors from the fact base.

(defrule inverter
  (or (ptrans ?id1 ?gate vdd ?a ?x1 ?y1)
```

```
      (ptrans ?id1 ?gate ?a vdd ?x1 ?y1))
  (or (ntrans ?id2 ?gate gnd ?a ?x2 ?y2)
      (ntrans ?id2 ?gate ?a gnd ?x2 ?y2))
  ?p <- (ptrans ?id1 $?)
  ?n <- (ntrans ?id2 $?)
=>
  (bind ?xa (/ (+ ?x1 ?x2) 2))
  (bind ?ya (/ (+ ?y1 ?y2) 2))
  (retract ?p ?n)
  (fprintout component "(inv "?gate" "?a" "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; tgate writes a tgate, averaging the positions, and retracts
;; the component transistors from the fact base.

(defrule tgate
  (or (ptrans ?id1 ?gate ?a ?b ?x1 ?y1)
      (ptrans ?id1 ?gate ?b ?a ?x1 ?y1))
  (or (ntrans ?id2 ?h ?a ?b ?x2 ?y2)
      (ntrans ?id2 ?h ?b ?a ?x2 ?y2))
  ?p <- (ptrans ?id1 $?)
  ?n <- (ntrans ?id2 $?)
=>
  (bind ?xa (/ (+ ?x1 ?x2) 2))
  (bind ?ya (/ (+ ?y1 ?y2) 2))
  (bind ?place (gensym))
  (retract ?p ?n)
  (fprintout component "(tgate "?place" "?gate" "?h" "?a)
  (fprintout component " "?b" "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; nand2 writes a nandgate, averaging the positions, and
;; retracts the component transistors from the fact base.

(defrule nand2
  (or (ptrans ?id1 ?a vdd ?o ?x1 ?y1)
      (ptrans ?id1 ?a ?o vdd ?x1 ?y1))
  (or (ntrans ?id2 ?a ?x gnd ?x2 ?y2)
      (ntrans ?id2 ?a gnd ?x ?x2 ?y2))
  (or (ptrans ?id3 ?b vdd ?o ?x3 ?y3)
      (ptrans ?id3 ?b ?o vdd ?x3 ?y3))
  (or (ntrans ?id4 ?b ?x ?o ?x4 ?y4)
      (ntrans ?id4 ?b ?o ?x ?x4 ?y4))
  ?p1 <- (ptrans ?id1 $?)
  ?n1 <- (ntrans ?id2 $?)
  ?p2 <- (ptrans ?id3 $?)
  ?n2 <- (ntrans ?id4 $?)
  (test (neq ?id1 ?id3))
  (test (neq ?id2 ?id4))
  (test (neq gnd ?o))
```

```
  (test (neq vdd ?o))
  (test (neq ?x ?o))
=>
  (bind ?xa (/ (+ ?x1 ?x2 ?x3 ?x4) 4))
  (bind ?ya (/ (+ ?y1 ?y2 ?y3 ?y4) 4))
  (bind ?place (gensym))
  (retract ?p1 ?n1 ?p2 ?n2)
  (fprintout component "(nand2 "?place" "?a" "?b" "?o)
  (fprintout component " "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; nor2 writes a norgate, averaging the positions, and
;; retracts the component transistors from the fact base.

(defrule nor2
  (or (ptrans ?id1 ?a vdd ?x ?x1 ?y1)
      (ptrans ?id1 ?a ?x vdd ?x1 ?y1))
  (or (ntrans ?id2 ?a ?o gnd ?x2 ?y2)
      (ntrans ?id2 ?a gnd ?o ?x2 ?y2))
  (or (ptrans ?id3 ?b ?x ?o ?x3 ?y3)
      (ptrans ?id3 ?b ?o ?x ?x3 ?y3))
  (or (ntrans ?id4 ?b gnd ?o ?x4 ?y4)
      (ntrans ?id4 ?b ?o gnd ?x4 ?y4))
  ?p1 <- (ptrans ?id1 $?)
  ?n1 <- (ntrans ?id2 $?)
  ?p2 <- (ptrans ?id3 $?)
  ?n2 <- (ntrans ?id4 $?)
  (test (neq ?id1 ?id3))
  (test (neq ?id2 ?id4))
  (test (neq gnd ?o))
  (test (neq vdd ?o))
  (test (neq ?x ?o))
=>
  (bind ?xa (/ (+ ?x1 ?x2 ?x3 ?x4) 4))
  (bind ?ya (/ (+ ?y1 ?y2 ?y3 ?y4) 4))
  (bind ?place (gensym))
  (retract ?p1 ?n1 ?p2 ?n2)
  (fprintout component "(nor2 "?place" "?a" "?b" "?o)
  (fprintout component " "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; close file closes outcomp1.clp, then saves the remaining
;; facts to the file comprem1.clp.
 (defrule close-file
  (declare (salience -99))
  ?i <- (close-file)
=>
  (retract ?i)
  (save-facts "comprem1.clp")
  (close component))
```

TR2.CLP, second CLIPS file for first level of extraction.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TR2.CLP  written by Capt Stuart Yarost as part of the
;; requirements for MS in Computer Engineering.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This file loads the facts from comprem1.clp, then builds
;; up the following components from the n and p type
;; transistors:
;;   clocked-inverter
;; As it writes these components to the file outcomp1.clp, it
;; retracts its component transistors from the database.  When
;; it is complete, it writes the remaining facts to the file
;; comprem1.clp.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; load-facts loads the facts from comprem1.clp.

(defrule load-facts
  ?i <- (initial-fact)
=>
  (load-facts "comprem1.clp")
  (retract ?i)
  (assert (open-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; open-file opens outcomp1.clp as a append only file.

(defrule open-file
  (declare (salience 101))
  ?i <- (open-file)
=>
  (open "outcomp1.clp" component "a")
  (retract ?i)
  (assert (close-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; clk_inv writes an clocked-inverter, averaging the
;; positions, and retracts the component transistors from the
;; fact base.

(defrule clk_inv
  (or (ptrans ?id1 ?p1 ?x ?drain ?x1 ?y1)
      (ptrans ?id1 ?p1 ?drain ?x ?x1 ?y1))
  (or (ntrans ?id2 ?gate gnd ?y ?x2 ?y2)
      (ntrans ?id2 ?gate ?y gnd ?x2 ?y2))
  (or (ptrans ?id3 ?gate ?x vdd ?x3 ?y3)
      (ptrans ?id3 ?gate vdd ?x ?x3 ?y3))
  (or (ntrans ?id4 ?p2 ?y ?drain ?x4 ?y4)
      (ntrans ?id4 ?p2 ?drain ?y ?x4 ?y4))
  ?pt1 <- (ptrans ?id1 $?)
```

```
   ?nt1 <- (ntrans ?id2 $?)
   ?pt2 <- (ptrans ?id3 $?)
   ?nt2 <- (ntrans ?id4 $?)
   (test (neq ?id1 ?id3))
   (test (neq ?id2 ?id4))
=>
   (bind ?xa (/ (+ ?x1 ?x2 ?x3 ?x4) 4))
   (bind ?ya (/ (+ ?y1 ?y2 ?y3 ?y4) 4))
   (retract ?pt1 ?nt1 ?pt2 ?nt2)
   (fprintout component "(clk_inv "?p1" "?p2" "?gate" "?drain)
   (fprintout component " "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; close file closes outcomp1.clp, then saves the remaining
;; facts to the file comprem1.clp.

(defrule close-file
   (declare (salience -99))
   ?i <- (close-file)
=>
   (retract ?i)
   (save-facts "comprem1.clp")
   (close component))
```

TR3.CLP, first CLIPS file for second level of extraction.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TR3.CLP  written by Capt Stuart Yarost as part of the
;; requirements for MS in Computer Engineering.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This file builds up the following components from the n and
;; p type transistors as well as lower order logic gates:
;;   buffer
;;   mux
;;   xnor (2 input)
;;   xor  (2 input)
;;   d-flipflop
;; As it writes these components to the file outcomp2.clp, it
;; retracts its component transistors from the database.  When
;; it is complete, it writes the remaining facts to the file
;; comprem2.clp.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; load-facts loads the facts from otcomp1.clp and
;; comprem1.clp.

(defrule load-facts
  ?i <- (initial-fact)
=>
  (load-facts "outcomp1.clp")
  (load-facts "comprem1.clp")
  (retract ?i)
  (assert (open-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; open-file opens outcomp2.clp as a write only file.

(defrule open-file
  (declare (salience 101))
  ?i <- (open-file)
=>
  (open "outcomp2.clp" component "w")
  (retract ?i)
  (assert (close-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; buffer writes a buffer, averaging the positions, and
;; retracts the ;; component lower level logic gates from the
;; fact base.

(defrule buffer
  ?inv1 <- (inv ?a ?b ?x1 ?y1)
  ?inv2 <- (inv ?b ?c ?x2 ?y2)
  (not (~inv ?b $?))
  (not (~inv ? ?b $?))
```

```
   (not (~inv ? ? ?b $?))
   (not (~inv ? ? ? ?b $?))
   (not (~inv ? ? ? ? ?b $?))
=>
   (bind ?xa (/ (+ ?x1 ?x2) 2))
   (bind ?ya (/ (+ ?y1 ?y2) 2))
   (retract ?inv1 ?inv2)
   (fprintout component "(buffer "?a" "?c" "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; mux writes a mux, averaging the positions, and retracts the
;; component lower level logic gates from the fact base.

(defrule mux
   (or (tgate ?id1 ?g ?h ?a ?c ?x1 ?y1)
       (tgate ?id1 ?g ?h ?c ?a ?x1 ?y1))
   (or (tgate ?id2 ?h ?g ?c ?b ?x2 ?y2)
       (tgate ?id2 ?h ?g ?b ?c ?x2 ?y2))
   (test (neq ?id1 ?id2))
   ?t1 <- (tgate ?id1 $?)
   ?t2 <- (tgate ?id2 $?)
=>
   (bind ?xa (/ (+ ?x1 ?x2) 2))
   (bind ?ya (/ (+ ?y1 ?y2) 2))
   (retract ?t1 ?t2)
   (fprintout component "(mux "?id1" "?g" "?h" "?a" "?b" "?c)
   (fprintout component " "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; xnor writes a xnor, averaging the positions, and retracts
;; the component lower level logic gates from the fact base.

(defrule xnor
   ?inv1 <- (inv ?a ?an ?x1 ?y1)
   ?inv2 <- (inv ?b ?bn ?x2 ?y2)
   (or (tgate ?id1 ?b ?bn ?an ?xor ?x3 ?y3)
       (tgate ?id1 ?b ?bn ?xor ?an ?x3 ?y3))
   (or (tgate ?id2 ?bn ?b ?a ?xor ?x4 ?y4)
       (tgate ?id2 ?bn ?b ?xor ?a ?x4 ?y4))
   ?tg1 <- (tgate ?id1 $?)
   ?tg2 <- (tgate ?id2 $?)
   (test (neq ?a ?b))
=>
   (bind ?xa (/ (+ ?x1 ?x2 ?x3 ?x4) 4))
   (bind ?ya (/ (+ ?y1 ?y2 ?y3 ?y4) 4))
   (bind ?place (gensym))
   (retract ?inv1 ?inv2 ?tg1 ?tg2)
   (fprintout component "(xnor "?place" "?a" "?an" "?b" "?bn)
   (fprintout component " "?xor" "?xa" "?ya")"crlf))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; xor writes a xor, averaging the positions, and retracts the
;; component lower level logic gates from the fact base.

(defrule xor
  ?inv1 <- (inv ?a ?an ?x1 ?y1)
  ?inv2 <- (inv ?b ?bn ?x2 ?y2)
  (or (tgate ?id1 ?b ?bn ?an ?xor ?x3 ?y3)
      (tgate ?id1 ?b ?bn ?xor ?an ?x3 ?y3))
  (or (tgate ?id2 ?b ?bn ?a ?xor ?x4 ?y4)
      (tgate ?id2 ?b ?bn ?xor ?a ?x4 ?y4))
  ?tg1 <- (tgate ?id1 $?)
  ?tg2 <- (tgate ?id2 $?)
  (test (neq ?a ?b))
=>
  (bind ?xa (/ (+ ?x1 ?x2 ?x3 ?x4) 4))
  (bind ?ya (/ (+ ?y1 ?y2 ?y3 ?y4) 4))
  (bind ?place (gensym))
  (retract ?inv1 ?inv2 ?tg1 ?tg2)
  (fprintout component "(xor "?place" "?a" "?an" "?b" "?bn)
  (fprintout component " "?xor" "?xa" "?ya")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; dff writes a d-type flipflop, averaging the positions, and
;; retracts the component lower level logic gates from the
;; fact base.

(defrule dff
  (or (tgate ?id1 ?p1 ?p2 ?d ?x ?x1 ?y1)
      (tgate ?id1 ?p1 ?p2 ?x ?d ?x1 ?y1))
  ?ci <- (clk_inv ?p2 ?p1 ?g ?x ?x2 ?y2)
  ?inv <- (inv ?x ?g ?x3 ?y3)
  ?tg <- (tgate ?id1 $?)
=>
  (bind ?xa (/ (+ ?x1 ?x2 ?x3) 3))
  (bind ?ya (/ (+ ?y1 ?y2 ?y3) 3))
  (retract ?ci ?inv ?tg)
  (fprintout component "(dff "?p1" "?p2" "?d" "?g)
  (fprintout component " "?xa" "?ya")"crlf))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; close file closes outcomp2.clp, then saves the remaining
;; facts to the file comprem2.clp.

(defrule close-file
  (declare (salience -99))
  ?i <- (close-file)
=>
  (retract ?i)
  (save-facts "comprem2.clp")
  (close component))
```

FINDEXT.CLP, CLIPS file to find extreme x and y values.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; FINDEXT.CLP  written by Capt Stuart Yarost as part of the
;; requirements for MS in Computer Engineering.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This file finds the maximum and minimum x and y
;; coordinates, then inserts them as the first fact in the
;; file of components.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; open-file opens as write only, the file scaled.clp.

(defrule open-file
  (declare (salience 101))
  ?i <- (initial-fact)
=>
  (open "scaled.clp" component "w")
  (retract ?i)
  (assert (list limits))
  (assert (close-file)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; first-big-x asserts a number as the largest x.

(defrule first-big-x
  (declare (salience 100))
  ($?most ?x-val ?y-val)
  (not (big-x ?))
  (test (numberp ?x-val))
  (test (numberp ?y-val))
=>
  (assert (big-x ?x-val)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; big-x finds the largest x and asserts that fact

(defrule big-x
  (declare (salience 100))
  ?x1 <- (big-x ?x)
  ($?most ?x-val ?y-val)
  (test (numberp ?x-val))
  (test (numberp ?y-val))
  (test (> ?x-val ?x))
=>
  (retract ?x1)
  (assert (big-x ?x-val)) )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; first-small-x asserts a number as the smallest x.

(defrule first-small-x
  (declare (salience 100))
  ($?most ?x-val ?y-val)
  (not (small-x ?))
  (test (numberp ?x-val))
  (test (numberp ?y-val))
=>
  (assert (small-x ?x-val)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; small-x finds the smallest x and asserts that fact

(defrule small-x
  (declare (salience 100))
  ?x1 <- (small-x ?x)
  ($?most ?x-val ?y-val)
  (test (numberp ?x-val))
  (test (numberp ?y-val))
  (test (< ?x-val ?x))
=>
  (retract ?x1)
  (assert (small-x ?x-val)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; first-big-y asserts a number as the biggest y.

(defrule first-big-y
  (declare (salience 100))
  ($?most ?x-val ?y-val)
  (not (big-y ?))
  (test (numberp ?x-val))
  (test (numberp ?y-val))
=>
  (assert (big-y ?y-val)) )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; big-y finds the biggest y and asserts that fact

(defrule big-y
  (declare (salience 100))
  ?y1 <- (big-y ?y)
  ($?most ?x-val ?y-val)
  (test (numberp ?x-val))
  (test (numberp ?y-val))
  (test (> ?y-val ?y))
=>
  (retract ?y1)
  (assert (big-y ?y-val)) )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; first-small-y asserts a number as the smallest y.

(defrule first-small-y
  (declare (salience 100))
  ($?most ?x-val ?y-val)
  (not (small-y ?))
  (test (numberp ?x-val))
  (test (numberp ?y-val))
=>
  (assert (small-y ?y-val)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; small-y finds the smallest y and asserts that fact

(defrule small-y
  (declare (salience 100))
  ?y1 <- (small-y ?y)
  ($?most ?x-val ?y-val)
  (test (numberp ?x-val))
  (test (numberp ?y-val))
  (test (< ?y-val ?y))
=>
  (retract ?y1)
  (assert (small-y ?y-val)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Asserts a fact of the largest new x and y values.

(defrule assert-limits
  (declare (salience 50))
  ?lf <- (list limits)
  (big-x ?big-x)
  (small-x ?small-x)
  (big-y ?big-y)
  (small-y ?small-y)
=>
  (bind ?smallx  (trunc ?small-x))
  (bind ?bigx  (trunc ?big-x))
  (bind ?smally  (trunc ?s    -y))
  (bind ?bigy  (trunc ?b    ))
  (retract ?lf)
  (fprintout component "(range "?smally" "?bigx" "?smally)
  (fprintout component " "?bigy")"crlf))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; write-all writes the rest of the components into the file
;; scaled.clp after the range information.

(defrule write-all
  ?comp <- ($?most ?x-val ?y-val)
  (test (numberp ?x-val))
  (test (numberp ?y-val))
=>
  (retract ?comp)
  (fprintout component "("$?most" "(trunc ?x-val))
  (fprintout component " "(trunc ?y-val)")"crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; close-file closes the file scaled.clp

(defrule close-file
  (declare (salience -99))
  ?i <- (close-file)
=>
  (retract ?i)
  (close component))
```

Appendix B: Graphical Display Code

DRAWGATE, batch program to run "drawit".

```
#!/bin/csh -f
shelltool -Wp 0 0 -Ws 900 900 "drawit"
```

DRAWIT.C, program to display extracted circuit.

```
/*************************************************************
*    Date:  8 November 1989
*    Version: 1.0
*
*    Title: Graphical Display Routine
*    Filename: drawit.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description:
*         This routine takes a component net-list
*         produced by the extraction routines, and
*         displays graphically the circuit.
*    Passed Variables: Component type, node names,
*                      x and y coordinates.
*    Returns: None
*    Files Read: xxx.clp, where xxx is the filename of an
*                output file from the extraction process.
*    Hardware Input: Right mouse button.
*    Modules Called: buffer.c clk_inv.c dff.c inv.c mux.c
*                    nand2.c nor2.c ntrans.c ptrans.c
*                    tgate.c xnor.c xor.c
*    Files Written: none
*    Special Instructions : Must be run in the SunView
*                           environment.
*                           Run batch file "drawgate" to
*                           start the program.
*************************************************************/
#include <stdio.h>
#include <string.h>
#include <usercore.h>
#define max_buf 128
#define minview(n) (0.5 - ((n)/2))
#define maxview(n) (0.5 + ((n)/2))
FILE *fd;
char fname[14];
char buff[max_buf];
```

```
char tempbuf[max_buf];
char gatetype[20], gate1[20], gate2[20], gate3[20],
                   gate4[20], gate5[20], gate6[20],
                   junk[20], another[2];
int xpos, ypos, count, count2, butt, xmax, xmin, ymax, ymin;
int pixwindd();
float xrange, yrange, xperc, yperc;
struct vwsurf vsurf = DEFAULT_VWSURF(pixwindd);
struct vwsurf vsurf2 = DEFAULT_VWSURF(pixwindd);

main()
{
  another[0] = 'y';
  while (another[0] == 'y')
     {
     do
       {
       printf("What file contains the circuit?\n");
       scanf("%s", fname);
       fd=fopen(fname,"r");
       } while (fd==NULL);
       fgets(buff,max_buf,fd);
       count2 =1;
       count = 1;
       while(buff[count]!=')')
        {
        tempbuf[count-1] = buff[count];
        count++;
         }
     tempbuf[count-1]=' ';
     tempbuf[count]=10;
     tempbuf[count+1]=0;
     sscanf(tempbuf, "%s %d %d %d %d", gatetype, &xmin,
                      &xmax, &ymin, &ymax);
     printf("The coordinate limits are as follows:\n");
     printf("xmin = %d\n",xmin);
     printf("xmax = %d\n",xmax+5000);
     printf("ymin = %d\n",ymin);
     printf("ymax = %d\n\n",ymax+5000);
     printf("What part of the screen");
     printf(" do you wish to display?\n");
     printf("Please enter \"xmin xmax ymin ymax\"):
     printf(" without qoutes.\n");
     scanf("%d %d %d %d", &xmin, &xmax, &ymin, &ymax);
     xrange = xmax - xmin;
     yrange = ymax - ymin;
     if (xrange > yrange)
         {
         xperc = 1;
         yperc = yrange / xrange;
```

```
    }
else
    {
    yperc = 1;
    xperc = xrange / yrange;
    }
vsurf = vsurf2;
initialize_core(DYNAMICB, NOINPUT, TWOD);
initialize_view_surface(&vsurf, FALSE);
select_view_surface(&vsurf);
set_ndc_space_2(1.0, 1.0);
set_viewport_2(minview(xperc), maxview(xperc),
               minview(yperc), maxview(yperc));
set_window((float)xmin, (float)xmax, (float)ymin,
          (float)ymax);
set_output_clipping(TRUE);
set_window_clipping(FALSE);
create_retained_segment(count2);
move_abs_2((float)xmin, (float)ymin);
line_rel_2(xrange, 0.0);
line_rel_2(0.0, yrange);
line_rel_2(-1.0*(xrange), 0);
line_rel_2(0.0, -1.0*(yrange));
close_retained_segment();
count2++;
initialize_device(BUTTON, 3);
initialize_device(LOCATOR, 1);
set_echo_surface(LOCATOR, 1, &vsurf);
set_echo_surface(BUTTON, 3, &vsurf);
set_echo(LOCATOR, 1, 1);

set_charprecision(CHARACTER);
set_text_index(1);

while(fgets(buff,max_buf,fd) != NULL)
   {

   count = 1;
   count2++;
   while(buff[count]!=')')
        {
        tempbuf[count-1] = buff[count];
        count++;
        }
   tempbuf[count-1]=' ';
   tempbuf[count]=10;
   tempbuf[count+1]=0;

sscanf(tempbuf, "%s", gatetype);
```

```
if (!(strcmp(gatetype, "ntrans")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
            junk, gate1, gate2, gate3, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    ntrans((float) xpos, (float) ypos, gate1, gate2,
           gate3);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "ptrans")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
           junk, gate1, gate2, gate3, &xpos, &ypos);
    set_image_transformation type(NONE);
    create_retained_segment(count2);
    ptrans((float) xpos, (float) ypos, gate1, gate2,
           gate3);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "inv")))
    {
    sscanf(tempbuf, "%s %s %s %d %d", gatetype,
           gate1, gate2, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    inv((float) xpos, (float) ypos, gate1, gate2);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "tgate")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %s %d %d",
           gatetype, junk, gate1, gate2, gate3,
           gate4, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    tgate((float) xpos, (float) ypos, gate1, gate2,
           gate3, gate4);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "nand2")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
           junk, gate1, gate2, gate3, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
```

```
    nand2((float) xpos, (float) ypos, gate1, gate2,
           gate3);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "nor2")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
           junk, gate1, gate2, gate3, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    nor2((float) xpos, (float) ypos, gate1, gate2,
          gate3);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "clk_inv")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
           gate1, gate2, gate3, gate4, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    clk_inv((float) xpos, (float) ypos, gate1, gate2,
             gate3, gate4);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "buffer")))
    {
    sscanf(tempbuf, "%s %s %s %d %d", gatetype,
           gate1, gate2, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    buffer((float) xpos, (float) ypos, gate1, gate2);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "mux")))
    {
    sscanf(tempbuf, "%s %s %s %s %s %s %s %d %d",
           gatetype, junk, gate1, gate2, gate3,
           gate4, gate5, &xpos, &ypos);
    set_image_transformation_type(NONE);
    create_retained_segment(count2);
    mux((float) xpos, (float) ypos, gate1, gate2,
         gate3, gate4, gate5);
    close_retained_segment();
    }

else if (!(strcmp(gatetype, "xnor")))
```

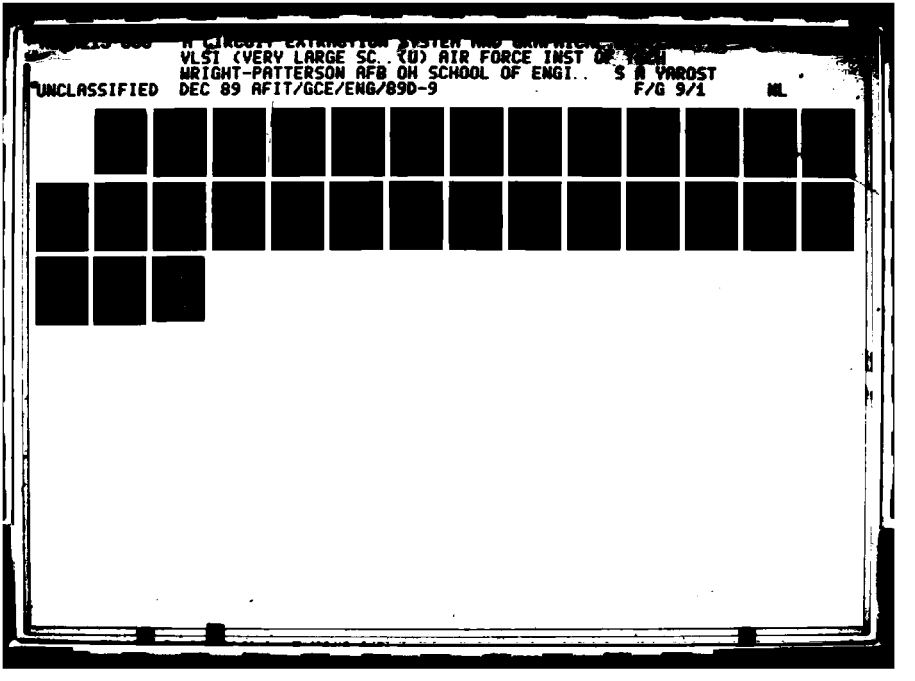
```
            {
            sscanf(tempbuf, "%s %s %s %s %s %s %s %d %d",
                   gatetype, junk, gate1, gate2, gate3,
                   gate4, gate5, &xpos, &ypos);
            set_image_transformation_type(NONE);
            create_retained_segment(count2);
            xnor((float) xpos, (float) vpos, gate1, gate2,
                  gate3, gate4, gate5);
            close_retained_segment();
            }

        else if (!(strcmp(gatetype, "xor")))
            {
            sscanf(tempbuf, "%s %s %s %s %s %s %s %d %d",
                   gatetype, junk, gate1, gate2, gate3,
                   gate4, gate5, &xpos, &ypos);
            set_image_transformation_type(NONE);
            create_retained_segment(count2);
            xor((float) xpos, (float) ypos, gate1, gate2,
                 gate3, gate4, gate5);
            close_retained_segment();
            }

        else if (!(strcmp(gatetype, "dff")))
            {
            sscanf(tempbuf, "%s %s %s %s %s %d %d", gatetype,
                   gate1, gate2, gate3, gate4, &xpos, &ypo );
            set_image_transformation_type(NONE);
            create_retained_segment(count2);
            dff((float) xpos, (float) ypos, gate1, gate2,
                 gate3, gate4);
            close_retained_segment();
            }
        }

    fclose(fd);

    butt = 0;
    while (butt == 0) await_any_button(1,&butt);

    terminate_device(BUTTON,3);
    terminate_device(LOCATOR,1);
    deselect_view_surface(&vsurf);
    terminate_core();

    printf("\n\nWould you like to do another file or");
    printf(" view, y or n?  ");
    scanf("%1s", another);
    }
}
```

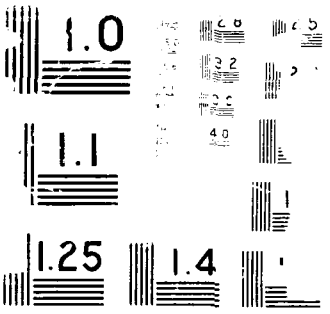
```
     BUFFER.C, function to display buffer.
/*************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: buffer.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a buffer.
*    Passed Variables: Component type, node names,
*                          x and y coordinates.
*    Calling Modules: Drawit.c
*************************************************************/

#include <usercore.h>
#include <string.h>

static float bufferx[] = {1000.0, 400.0, -400.0, -1000.0,
                          0.0, -400.0, 400.0, 0.0};

static float buffery[] = {500.0, 0.0, 0.0, 500.0, -500,
                          0.0, 0.0, -500.0};

buffer(x0, y0, gate1, gate2)
float x0, y0;
char *gate1, *gate2;
{
    move_abs_2(x0+2000.0, y0+500.0);
    polyline_rel_2(bufferx, buffery, 8);
    set_charspace(50.0);
    set_char ize(160.0, 160.0);
    move_ _2(-210.0*(strlen(gate1)+0.5), 670.0);
    text(gate1);
    move_abs_2(x0+3100.0, y0+1170.0);
    text(gate2);
}
```

CLK_INV.C, function to display a clocked inverter.

```
/*************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: clk_inv.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a clocked
*                 inverter.
*    Passed Variables: Component type, node names,
*                      x and y coordinates.
*    Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>
static float cinvx[] = {500.0, 29.3, 70.7, 70.7, 29.3, 200.0,
                        -200.0, -29.3, -70.7, -70.7, -29.3,
                        -500.0, 0.0, -200.0, 200.0, 0.0};
static float cinvy[] = {250.0, 70.7, 29.3, -29.3, -70.7, 0.0,
                        0.0, -70.7, -29.3, 29.3, 70.7, 250.0,
                        -250, 0.0, 0.0, -250.0};
static float cinvx2[] = {0.0, 17.7, 7.3, -7.3, -17.7, 0.0,
                         0.0, -17.7, -7.3, 7.3, 17.7};
static float cinvy2[] = {-375.0, -7.3, -17.7, -17.7, -7.3,
                         -75.0, 75.0, 7.3, 17.7, 17.7, 7.3};
clk_inv(x0, y0, gate1, gate2, gate3, gate4)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4;
{
    move_abs_2(x0+1000.0, y0+250.0);
    polyline_rel_2(cinvx, cinvy, 16);
    move_abs_2(x0+1250.0, y0+750.0);
    polyline_rel_2(cinvx2, cinvy2, 11);
    set_charspace(25.0);
    set_charsize(80.0, 80.0);
    move_abs_2(x0+1250.0+(-52.5*(strlen(gate1))), (y0+150.0));
    text(gate1);
    move_abs_2(x0+1250.0+(-52.5*(strlen(gate2))), (y0+850.0));
    text(gate2);
    move_abs_2(x0+1000.0+(-105.0*(strlen(gate3)+0.5)),
              (y0+585.0));
    text(gate3);
    move_abs_2(x0+1750.0, y0+585.0);
    text(gate4);
}
```

```
     DFF.C, function to display a D type Flip Flop.
/*********************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: dff.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: Displays a D type flip-flop.
*    Passed Variables: Gatetype, node names, coordinates.
*    Calling Modules: Drawit.c
*********************************************************/
#include <usercore.h>
#include <string.h>
static float dffx[] = {500.0, 0.0, 0.0, 500.0, 0.0, 400.0,
  -400.0, 0.0, -500.0, 0.0, 0.0, -500.0, 0.0, -400.0, 400.0,
  0.0};
static float dffy[] = {0.0, -50.0, 50.0, 0.0, 500.0, 0.0, 0.0,
  500.0, 0.0, 100.0, -100.0, 0.0, -500, 0.0, 0.0, -500.0};
dff(x0, y0, gate1, gate2, gate3, gate4)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4;
{
 move_abs_2(x0+2000.0, y0+500.0);
 polyline_rel_2(dffx, dffy, 16);
 set_charspace(50.0);
 set_charsize(160.0, 160.0);
 move_abs_2(x0+2500.0+(-105.0*(strlen(gate1))),y0+1700.0);
 text(gate1);
 move_abs_2(x0+2500.0+(-105.0*(strlen(gate2))), y0+280.0);
 text(gate2);
 move_abs_2(x0+2000.0+(-210.0*(strlen(gate3)+0.5)),y0+1170.0);
 text(gate3);
 move_abs_2(x0+3100.0, y0+1170.0);
 text(gate4);
 move_abs_2(x0+2180.0, y0+1000.0);
 set_charspace(1.0);
 text("DFF");
 set_charspace(30.0);
 set_charsize(120.0, 120.0);
 move_abs_2(x0+2010.0, y0+1000.0);
 text("D");
 move_abs_2(x0+2790.0, y0+1000.0);
 text("Q");
 move_abs_2(x0+2300.0, y0+1350.0);
 text("P1");
 move_abs_2(x0+2290.0, y0+630.0);
 text("P2");
}
```


VLSI (VERY LARGE SC.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S A VAROST
UNCLASSIFIED DEC 89 AFIT/GCE/ENG/89D-9 F/G 9/1 NL


1.0
1.1
1.25
1.4

```
     INV.C, function to display an inverter.
/*****************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: inv.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays an inverter.
*    Passed Variables: Component type, node names,
*                         x and y coordinates.
*    Calling Modules: Drawit.c
*****************************************************************/
#include <usercore.h>
#include <string.h>

static float invx[] = {500.0, 29.3, 70.7, 70.7, 29.3, 250.0,
                       -250.0, -29.3, -70.7, -70.7, -29.3,
                       -500.0, 0.0, -250.0, 250.0, 0.0,};
static float invy[] = {250.0, 70.7, 29.3, -29.3, -70.7, 0.0,
                       0.0, -70.7, -29.3, 29.3, 70.7, 250.0,
                       -250.0, 0.0, 0.0, -250.0};

inv(x0, y0, gate1, gate2)
float x0, y0;
char *gate1, *gate2;
{
    move_abs_2(x0+1000.0, y0+250.0);
    polyline_rel_2(invx, invy, 16);
    set_charspace(25.0);
    set_charsize(80.0, 80.0);
    move_rel_2(-105.0*(strlen(gate1)+0.5), 335.0);
    text(gate1);
    move_abs_2(x0+1750.0, y0+585.0);
    text(gate2);
}
```

```
     MUX.C, function to display a multiplexor.
 /*************************************************************
*     Date:  8 November 1989
*     Title: Graphical Display Routine
*     Filename: mux.c
*     Author: Capt Stuart Yarost
*     Project: Extraction System and Graphical Display
*     Operating System: Unix V4.3
*     Language:  C
*     Description: This function displays a multiplexor.
*     Passed Variables: Component type, node names,
*                          x and y coordinates.
*     Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>
static float muxx[] = {500.0, 0.0, 0.0, 500.0, 0.0, 400.0,
                       -400.0, 0.0, -500.0, 0.0, 0.0, -500.0,
                       0.0, -400.0, 400.0, 0.0, -400.0, 400.0,
                       0.0};
static float muxy[] = {0.0, -50.0, 50.0, 0.0, 500.0, 0.0,
                       0.0, 500.0, 0.0, 100.0, -100.0, 0.0,
                       -300, 0.0, 0.0, -400.0, 0.0, 0.0,
                       -300.0};
mux(x0, y0, gate1, gate2, gate3, gate4, gate5)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4, *gate5;
{
  move_abs_2(x0+2000.0, y0+500.0);
  polyline_rel_2(muxx, muxy, 19);
  set_charspace(50.0);
  set_charsize(160.0, 160.0);
  move_abs_2(x0+2500.0+(-105.0*(strlen(gate1))), y0+1700.0);
  text(gate1);
  move_abs_2(x0+2500.0+(-105.0*(strlen(gate2))), y0+280.0);
  text(gate2);
  move_abs_2(x0+2000.0+(-210.0*(strlen(gate3)+0.5)),
             y0+1370.0);
  text(gate3);
  move_abs_2(x0+2000.0+(-210.0*(strlen(gate4)+0.5)),
             y0+970.0);
  text(gate4);
  move_abs_2(x0+3100.0, y0+1170.0);
  text(gate5);
  move_abs_2(x0+2180.0, y0+1000.0);
  set_charspace(1.0);
  text("MUX");
}
```

```
     NAND2.C, function to display a NAND gate.
/*************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: nand2.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a NAND gate.
*    Passed Variables: Component type, node names,
*                      x and y coordinates.
*    Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>

static float nandx[] = {500.0, 95.65, 81.1, 54.2, 19.05, 29.3,
                        70.7, 70.7, 29.3, 200, -200, -29.3,
                        -70.7, -70.7, -29.3, -19.05, -54.2,
                        -81.1, -95.65, -500.0, 0.0, -200.0,
                        200.0, 0.0, -200.0, 200.0, 0.0};

static float nandy[] = {0.0, -19.05, -54.2, -81.1, -95.65,
                        70.7, 29.3, -29.3, -70.7, 0.0, 0.0,
                        -70.7, -29.3, 29.3, 70.7, -95.65,
                        -81.1, -54.2, -19.05, 0.0, 100.0, 0.0,
                        0.0, 300.0, 0.0, 0.0, 100.0};

nand2(x0, y0, gate1, gate2, gate3)
float x0, y0;
char *gate1, *gate2, *gate3;
{
    move_abs_2(x0, y0);
    move_rel_2(1000.0, 750.0);
    polyline_rel_2(nandx, nandy, 27);
    set_charspace(25.0);
    set_charsize(80.0, 80.0);
    move_rel_2(-105.0*(strlen(gate1)+0.5), -30.0);
    text(gate1);
    move_abs_2((-105.0*(strlen(gate2)+0.5)+x0+1000),420.0+y0);
    text(gate2);
    move_abs_2(x0+1980.0, y0+580.0);
    text(gate3);
}
```

```
     NOR2.C, function to display a NOR gate.
/*************************************************************
*     Date:  8 November 1989
*     Title: Graphical Display Routine
*     Filename: nor2.c
*     Author: Capt Stuart Yarost
*     Project: Extraction System and Graphical Display
*     Operating System: Unix V4.3
*     Language:  C
*     Description: This function displays a NOR gate.
*     Passed Variables: Component type, node names,
*                          x and y coordinates.
*     Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>
static float norx[] = {98.0, 97.1, 95.2, 92.4, 88.7, 84.2,
                       78.8, 72.7, 29.3, 70.7, 70.7, 29.3,
                       200, -200, -29.3, -70.7, -70.7, -29.3,
                       -72.7, -78.8, -84.2, -88.7, -92.4,
                       -95.2, -97.1, -98.0};
static float nory[] = {-4.8, -14.4, -23.8, -33.1, -42.0,
                       -50.5, -58.5, -65.9, 70.7, 29.3, 29.3,
                       -70.7, 0.0, 0.0, -70.7, -29.3, 29.3,
                       70.7, -65.9, -58.5, -50.5, -42.0,
                       -33.1, -23.8, -14.4,-4.8};
static float norx2[] = {25.3, 18.2, -300.0, 300.0, 11.0, 3.7,
                        -3.7, -11.0, -300.0, 300.0, -18.2,
                        -25.3};
static float nory2[] = {-70.7, -72.8, 0.0, 0.0, -74.3, -75.0,
                        -75.0, -74.3, 0.0, 0.0, -72.8, -70.7};

nor2(x0, y0, gate1, gate2, gate3)
float x0, y0;
char *gate1, *gate2, *gate3;
{
   move_abs_2(x0+1000, y0+750);
   polyline_rel_2(norx, nory, 26);
   move_abs_2(x0+1000, y0+750);
   polyline_rel_2(norx2, nory2, 12);
   set_charspace(25.0);
   set_charsize(80.0, 80.0);
   move_abs_2((-105.0*(strlen(gate1)+0.5)+x0+1040),665.0+y0);
   text(gate1);
   move_abs_2((-105.0*(strlen(gate2)+0.5)+x0+1040), 390.0+y0);
   text(gate2);
   move_abs_2(x0+1940.0, y0+560.0);
   text(gate3);
}
```

```
     NTRANS.C, function to display a N type transistor.
/*************************************************************
*     Date:  8 November 1989
*     Title: Graphical Display Routine
*     Filename: ntrans.c
*     Author: Capt Stuart Yarrost
*     Project: Extraction System and Graphical Display
*     Operating System: Unix V4.3
*     Language:  C
*     Description: This function displays a N-type
*                       transistor.
*     Passed Variables: Component type, node names,
*                            x and y coordinates.
*     Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>

static float ntranx[] = {0.0, -40.0, 0.0, 40.0, 0.0};
static float ntrany[] = {80.0, 0.0, 80.0, 0.0, 80.0};
static float ntranx2[] = {0.0, -60.0, 60.0 ,0.0};
static float ntrany2[] = {40.0, 0.0, 0.0, 40.0};

ntrans(x0, y0, gate1, gate2, gate3)
float x0, y0;
char *gate1, *gate2, *gate3;
{
    move_abs_2(x0+200.0, y0+70.0);
    polyline_rel_2(ntranx, ntrany, 5);
    move_abs_2(x0-150.0, y0+150.0);
    polyline_rel_2(ntranx2, ntrany2, 4);
    set_charspace(5.0);
    set_charsize(20.0, 20.0);
    move_abs_2(x0+1[illegible]-25.0*(strlen(gate1)+0.5), y0+210);
    text(gate1);
    move_abs_2(x0+20[illegible]-12.5*(strlen(gate2)), y0+330);
    text(gate2);
    move_abs_2(x0+200-12.5*(strlen(gate3)), y0+40);
    text(gate3);
}
```

PTRANS.C, function to display a P type transistor.

```
/*************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: ptrans.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a p-type
*                    transistor.
*    Passed Variables: Component type, node names,
*                         x and y coordinates.
*    Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>

static float ptranx[] = {0.0, -40.0, 0.0, 40.0, 0.0};
static float ptrany[] = {80.0, 0.0, 80.0, 0.0, 80.0};
static float ptranx2[] = {0.0, -2.9, -7.1, -7.1, -2.9, -60.0,
                          60.0 , 2.9, 7.1, 7.1, 2.9, 0.0};
static float ptrany2[] = {40.0, -7.1, -2.9, 2.9, 7.1, 0.0,
                          0.0, 7.1, 2.9, -2.9, -7.1, 40.0};

ptrans(x0, y0, gate1, gate2, gate3)
float x0, y0;
char *gate1, *gate2, *gate3;
{
    move_abs_2(x0+200.0, y0+70.0);
    polyline_rel_2(ptranx, ptrany, 5);
    move_abs_2(x0+150.0, y0+150.0);
    polyline_rel_2(ptranx2, ptrany2, 12);
    set_charspace(5.0);
    set_charsize(20.0, 20.0);
    move_abs_2(x0+130-25.0*(strlen(gate1)+0.5), y0+210);
    text(gate1);
    move_abs_2(x0+200-12.5*(strlen(gate2)), y0+330);
    text(gate2);
    move_abs_2(x0+200-12.5*(strlen(gate3)), y0+40);
    text(gate3);
}
```

```
     TGATE.C, function to display a tgate.
/*****************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: tgate.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a tgate.
*    Passed Variables: Component type, node names,
*                      x and y coordinates.
*    Calling Modules: Drawit.c
*****************************************************/
#include <usercore.h>
#include <string.h>

static float tgatex[] = {250.0, 0.0, 200.0, -200.0, 0.0,
                         -500.0, 0.0, -200.0, 200.0, 0.0,
                         250.0, 0.0, 0.0, -17.7, -7.3, 7.3,
                         17.7, 0.0, 0.0, 17.7, 7.3, -7.3,
                         -17.7};
static float tgatey[] = {-250.0, 250.0, 0.0, 0.0, 250.0,
                         -500.0, 250.0, 0.0, 0.0, 250.0,
                         -250.0, -250.0, 250.0, 7.3, 17.7,
                         17.7, 7.3, 225.0, -225.0, -7.3,
                         -17.7, -17.7, -7.3};

tgate(x0, y0, gate1, gate2, gate3, gate4)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4;
{
  move_abs_2(x0+1250.0, y0+500.0);
  polyline_rel_2(tgatex, tgatey, 23);
  set_charspace(25.0);
  set_charsize(80.0, 80.0);
  move_abs_2(x0+1250.0+(-52.5*(strlen(gate1))), y0+840.0);
  text(gate1);
  move_abs_2(x0+1250.0+(-52.5*(strlen(gate2))), y0+140.0);
  text(gate2);
  move_abs_2(x0+1000.0+(-105.0*(strlen(gate3)+0.5)),y0+585.0);
  text(gate3);
  move_abs_2(x0+1540.0, y0+585.0);
  text(gate4);
}
```

```
     XNOR.C, function to display a exclusive NOR gate.
/*************************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: xnor.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a X-NOR gate.
*    Passed Variables: gatetype, node names, coordinates.
*    Calling Modules: Drawit.c
*************************************************************/
#include <usercore.h>
#include <string.h>
static float xnorx[] = {196.0, 194.2, 190.4, 184.8, 177.4,
  168.4, 157.6, 145.4, 58.6, 141.4, 141.4, 58.6, 400, -400,
  -58.6, -141.4, -141.4, -58.6, -145.4, -157.6, -168.4,
  -177.4, -184.8, -190.4, -194.2,-196.0};
static float xnory[] = {-9.6, -28.8, -47.6, -66.2, -84.0,
  -101.0, -107.0, -131.8, 141.4, 58.6, -58.6, -141.4, 0.0,
  0.0, -141.4, -58.6, 58.6, 141.4, -131.8, -107.0, -101.0,
  -84.0, -66.2, -47.6, -28.8, -9.6};
static float xnorx2[] = {50.6, 36.4, -600.0, 600.0, 22.0, 7.4,
  -7.4, -22.0, -600.0, 600.0, -36.4, -50.6};
static float xnory2[] = {-141.4, -145.6, 0.0, 0.0, -148.6,
  -150.0, -150.0, -148.6, 0.0, 0.0,-145.6, -141.4};
static float xnorx3[] = {50.6, 36.4, 22.0, 7.4, -7.4, -22.0,
                         -36.4, -50.6};
static float xnory3[] = {-141.4, -145.6, -148.6, -150.0,
                         -150.0, -148.6, -145.6, -141.4};
xnor(x0, y0, gate1, gate2, gate3, gate4, gate5)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4, *gate5;
{
  move_abs_2(x0+2160, y0+1500);
  polyline_rel_2(xnorx, xnory, 26);
  move_abs_2(x0+2160, y0+1500);
  polyline_rel_2(xnorx3, xnory3, 8);
  move_abs_2(x0+2000, y0+1500);
  polyline_rel_2(xnorx2, xnory2, 12);
  set_charspace(50.0);
  set_charsize(160.0, 160.0);
  move_abs_2((-210.0*(strlen(gate1)+0.5)+x0+2080), 1330.0+y0);
  text(gate1);
  move_abs_2((-210.0*(strlen(gate3)+0.5)+x0+2080),780.0+y0);
  text(gate3);
  move_abs_2(x0+4040.0, y0+1120.0);
  text(gate5);
}
```

```
     XOR.C, function to display a exclusive OR gate.
/*********************************************************
*    Date:  8 November 1989
*    Title: Graphical Display Routine
*    Filename: xor.c
*    Author: Capt Stuart Yarost
*    Project: Extraction System and Graphical Display
*    Operating System: Unix V4.3
*    Language:  C
*    Description: This function displays a X-OR gate.
*    Passed Variables: Component type, node names,
*                        x and y coordinates.
*    Calling Modules: Drawit.c
*********************************************************/
#include <usercore.h>
#include <string.h>
static float xorx[] = {196.0, 194.2, 190.4, 184.8, 177.4,
   168.4, 157.6, 145.4, 400, -400, -145.4, -157.6, -168.4,
   -177.4, -184.8,-190.4, -194.2, -196.0};
static float xory[] = {-9.6, -28.8, -47.6, -66.2, -84.0,
   -101.0, -107.0, -131.8, 0.0, 0.0, -131.8, -107.0, -101.0,
   -84.0, -66.2,-47.6, -28.8, -9.6};
static float xorx2[] = {50.6, 36.4, -600.0, 600.0, 22.0, 7.4,
   -7.4, -22.0, -600.0, 600.0, -36.4, -50.6};
static float xory2[] = {-141.4, -145.6, 0.0, 0.0, -148.6,
                        -150.0, -150.0, -148.6, 0.0, 0.0,
                        -145.6, -141.4};
static float xorx3[] = {50.6, 36.4, 22.0, 7.4, -7.4, -22.0,
                        -36.4, -50.6};
static float xory3[] = {-141.4, -145.6, -148.6, -150.0,
                        -150.0, -148.6, -145.6, -141.4};
xor(x0, y0, gate1, gate2, gate3, gate4, gate5)
float x0, y0;
char *gate1, *gate2, *gate3, *gate4, *gate5;
{
  move_abs_2(x0+2160, y0+1500);
  polyline_rel_2(xorx, xory, 18);
  move_abs_2(x0+2160, y0+1500);
  polyline_rel_2(xorx3, xory3, 8);
  move_abs_2(x0+2000, y0+1500);
  polyline_rel_2(xorx2, xory2, 12);
  set_charspace(50.0);
  set_charsize(160.0, 160.0);
  move_abs_2((-210.0*(strlen(gate1)+0.5)+x0+2080), 1330.0+y0);
  text(gate1);
  move_abs_2((-210.0*(strlen(gate3)+0.5)+x0+2080),780.0+y0);
  text(gate3);
  move_abs_2(x0+3700.0, y0+1120.0);
  text(gate5);
}
```

Appendix B: Graphical Display Code

DRAWTEST, batch program to run "testgate.c".

```
#!/bin/csh -f
shelltool -Wp 0 0 -Ws 900 900 "testgate"
```

TESTGATE, program to test functions of "drawit.c".

```
/*************************************************************
*     Date:  8 November 1989
*     Title: Graphical Display Tester
*     Filename: testgate.c
*     Author: Capt Stuart Yarost
*     Project: Extraction System and Graphical Display
*     Operating System: Unix V4.3
*     Language:  C
*     Description:
*          This routine tests functions by drawing
*          components on the screen.
*
*     Passed Variables: Component type, node names
*     Returns: None
*     Files Read: none
*     Hardware Input: Right mouse button.
*     Modules Called: buffer.c clk_inv.c dff.c inv.c mux.c
*                     nand2.c nor2.c ntrans.c ptrans.c
*                     tgate.c xnor.c xor.c
*     Files Written: none
*     Special Instructions : Must be run in the SunView
*                            environment.
*                            Run batch file "drawtest" to
*                            start the program.
*
*************************************************************/

#include <stdio.h>
#include <string.h>
#include <usercore.h>


#define minview(n) (0.5 - ((n)/2))
#define maxview(n) (0.5 + ((n)/2))

char gatetype[20], another[2];
```

```
int butt, xmax, xmin, ymax, ymin;
int pixwindd();
float xrange, yrange, xperc, yperc;
struct vwsurf vsurf = DEFAULT_VWSURF(pixwindd);
struct vwsurf vsurf2 = DEFAULT_VWSURF(pixwindd);

main()
{
  another[0] = 'y';
  while (another[0] == 'y')
     {
     printf("What type component do you wish to test?\n");
     scanf("%s", gatetype);

     xmin = 0;
     xmax =10000;
     ymin = 0;
     ymax =10000;

     printf("The coordinate limits are as follows:\n");
     printf("xmin = %d\n",xmin);
     printf("xmax = %d\n ,xmax+5000);
     printf("ymin = %d\n",ymin);
     printf("ymax = %d\n\n",ymax+5000);

     printf("What part of tne screen ");
     printf("do you wish to display?\n");
     printf("Please enter \"xmin xmax ymin ymax\"");
     printf(" without qoutes.\n");
     scanf("%d %d %d %d", &xmin, &xmax, &ymin, &ymax);
     xrange = xmax - xmin;
     yrange = ymax - ymin;

     if (xrange > yrange)
          {
          xperc = 1;
          yperc = yrange / xrange;
          }
     else
          {
          yperc = 1;
          xperc = xrange / yrange;
          }
     vsurf = vsurf2;
     initialize_vwsurf(&vsurf, MINBUF, TWOD);
     set_viewsurface(&vsurf, FALSE);
     clear_viewsurface(&vsurf);
     set_viewspace_2(1.0, 1.0);
     set_viewport_2(minview(xperc), maxview(xperc),
                    minview(yperc), maxview(yperc));
```

```
set_window((float)xmin, (float)xmax,
           (float)ymin, (float)ymax);
set_output_clipping(TRUE);
set_window_clipping(FALSE);
create_retained_segment(1);
move_abs_2((float)xmin, (float)ymin);
line_rel_2(xrange, 0.0);
line_rel_2(0.0, yrange);
line_rel_2(-1.0*(xrange), 0);
line_rel_2(0.0, -1.0*(yrange));
close_retained_segment();
initialize_device(BUTTON, 3);
initialize_device(LOCATOR, 1);
set_echo_surface(LOCATOR, 1, &vsurf);
set_echo_surface(BUTTON, 3, &vsurf);
set_echo(LOCATOR, 1, 1);
set_charprecision(CHARACTER);
set_text_index(1);
out = 0;
if (!(strcmp(gatetype, "ntrans")))
    {
     set_image_transformation_type(NONE);
     create_retained_segment(2);
     ntrans(0.0, 0.0, "gate1", "gate2", "gate3");
     close_retained_segment();
    }
else if (!(strcmp(gatetype, "ptrans")))
    {
     set_image_transformation_type(NONE);
     create_retained_segment(2);
     ptrans(0.0, 0.0, "gate1", "gate2", "gate3");
     close_retained_segment();
    }
else if (!(strcmp(gatetype, "inv")))
    {
     set_image_transformation_type(NONE);
     create_retained_segment(2);
     inv(0.0, 0.0, "gate1", "gate2");
     close_retained_segment();
    }
else if (!(strcmp(gatetype, "tgate")))
    {
     set_image_transformation_type(NONE);
     create_retained_segment(2);
     tgate(0.0, 0.0, "gate1", "gate2", "gate3","gate4");

     close_retained_segment();
    }
else if (!(strcmp(gatetype, "nand2")))
    {
```

```
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    nand2(0.0, 0.0, "gate1", "gate2", "gate3");
    close_retained_segment();
   }
else if (!(strcmp(gatetype, "nor2")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    nor2(0.0, 0.0, "gate1", "gate2", "gate3");
    close_retained_segment();
   }
else if (!(strcmp(gatetype, "clk_inv")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    clk_inv(0.0, 0.0, "gate1", "gate2","gate3","gate4");

    close_retained_segment();
   }
else if (!(strcmp(gatetype, "buffer")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    buffer(0.0, 0.0, "gate1", "gate2");
    close_retained_segment();
   }
else if (!(strcmp(gatetype, "mux")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    mux(0.0, 0.0,"gate1","gate2","gate3","gate4",
        "gate5");
    close_retained_segment();
   }
else if (!(strcmp(gatetype, "xnor")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    xnor(0.0, 0.0, "gate1", "gate2", "gate3", "gate4",

         "gate5");
    close_retained_segment();
   }
else if (!(strcmp(gatetype, "xor")))
   {
    set_image_transformation_type(NONE);
    create_retained_segment(2);
    xor(0.0, 0.0, "gate1", "gate2", "gate3", "gate4",
        "gate5");
```

```
            close_retained_segment();
           }
         else if (!(strcmp(gatetype, "dff")))
           {
            set_image_transformation_type(NONE);
            create_retained_segment(2);
            dff(0.0, 0.0, "gate1", "gate2", "gate3", "gate4");

            close_retained_segment();
           }
         else
           {
           printf("That is not a testable component!\n");
           butt = 3;
         }
         while (butt == 0) await_any_button(1,&butt);
         terminate_device(BUTTON,3);
         terminate_device(LOCATOR,1);
         deselect_view_surface(&vsurf);
         terminate_core();
         printf("\n\nWould you like to test another");
         printf(" gate, y or n?  ");
         scanf("%1s", another);
         }
}
```

CIRCLEPT.C, program to generate moves to draw a circle.

```
/*************************************************************
*     Date:  8 November 1989
*     Title: Circle Coordinate Generator
*     Filename: circlept.c
*     Author: Capt Stuart Yarost
*     Project: Extraction System and Graphical Display
*     Operating System: Unix V4 3, DOS V3.3
*     Language:  C
*     Description:
*          Produces a list of x moves and y moves
*          necessary to draw a circle of x points and
*          y radius.  a and y are input by the user.
*************************************************************/

#include <stdio.h>
#include <math.h>

#define pi 3.141592654
#define arc 2*pi/number

int number, radius, count;
double xmov, ymov, xpos, ypos, oldx, oldy;

main()
{
     printf("How many points on the circle?\n");
     scanf("%d", &number);
     printf("What is the desired radius?\n");
     scanf("%d", &radius);
     oldx = -1.0;
     oldy = 0.0;
     for(count = 1; count <= number; count++)
        {
        xpos = cos(pi - (arc * count));
        ypos = sin(pi - (arc * count));
        xmov = (xpos - oldx)*radius;
        ymov = (ypos - oldy)*radius;
        oldx = xpos;
        oldy = ypos;
        printf("x move is :  %8.1f     y move is :", xmov);
        printf("   %8.1f\n", ymov);
        }
}
```

NEW.SIM

The following is the MEXTRA produced file of the clock generator circuit.

```
| units: 1     tech: cmos-pw format: UCB
p 128 OZ_pq1 Vdd 300 21596.3 -26700 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -28500 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -30300 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -32100 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -33900 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -42900 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -44700 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -46500 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -48300 -24000
p 128 OZ_pq1 Vdd 300 21596.3 -50100 -24000
p 53 OZ_pq2 Vdd 300 21596.3 76800 -29400
p 53 OZ_pq2 Vdd 300 21596.3 75000 -29400
p 53 OZ_pq2 Vdd 300 21596.3 73200 -29400
p 53 OZ_pq2 Vdd 300 21596.3 71400 -29400
p 53 OZ_pq2 Vdd 300 21596.3 69600 -29400
p 53 OZ_pq2 Vdd 300 21596.3 60600 -29400
p 53 OZ_pq2 Vdd 300 21596.3 58800 -29400
p 53 OZ_pq2 Vdd 300 21596.3 57000 -29400
p 53 OZ_pq2 Vdd 300 21596.3 55200 -29400
p 53 OZ_pq2 Vdd 300 21596.3 53400 -29400
C IZ_cap1 177 392 (900 -8100)
p 277 Vdd 233 300 13647.8 24450 -9900
p 329 277 Vdd 300 6748.75 18600 -11400
p 53 Vdd OZ_pq2 300 21596.3 53100 -17850
p 53 Vdd OZ_pq2 300 21596.3 54900 -17850
p 53 Vdd OZ_pq2 300 21596.3 56700 -17850
p 53 Vdd OZ_pq2 300 21596.3 58500 -17850
p 53 Vdd OZ_pq2 300 21596.3 60300 -17850
p 53 Vdd OZ_pq2 300 21596.3 69300 -17850
p 53 Vdd OZ_pq2 300 21596.3 71100 -17850
p 53 Vdd OZ_pq2 300 21596.3 72900 -17850
p 53 Vdd OZ_pq2 300 21596.3 74700 -17850
p 53 Vdd OZ_pq2 300 21596.3 76500 -17850
p 233 53 Vdd 300 21596.3 37200 -17100
p 233 53 Vdd 300 21596.3 39000 -17100
p 233 53 Vdd 300 21596.3 40800 -17100
p 233 53 Vdd 300 21596.3 42600 -17100
p 233 53 Vdd 300 21596.3 44400 -17100
p 277 Vdd 233 300 13647.8 24450 -12750
e 450 449 520 300 450 3300 -1350
e 498 520 GND 300 450 2700 -1350
e IZ_go GND 498 300 450 1500 -1350
e 424 453 GND 300 450 27600 -1650
e 453 GND 329 1200 450 25500 -2250
e 447 329 GND 1200 450 23550 -2250
```

```
e 329 GND 447 1200 450 21600 -2250
e 450 424 GND 300 450 17100 -2700
e 449 443 GND 600 450 5100 -2700
e 424 447 GND 1200 450 19500 -2850
e 177 444 GND 1200 450 10650 -3150
e 444 450 GND 1200 450 15000 -3300
e 443 177 GND 1200 450 7350 -3300
e 277 GND 233 300 13497.8 24300 -4800
p 128 Vdd OZ_pq1 300 21596.3 -30600 -12450
p 128 Vdd OZ_pq1 300 21596.3 -32400 -12450
p 128 Vdd OZ_pq1 300 21596.3 -34200 -12450
p 128 Vdd OZ_pq1 300 21596.3 -43200 -12450
p 128 Vdd OZ_pq1 300 21596.3 -45000 -12450
p 128 Vdd OZ_pq1 300 21596.3 -46800 -12450
p 128 Vdd OZ_pq1 300 21596.3 -48600 -12450
p 128 Vdd OZ_pq1 300 21596.3 -50400 -12450
p 128 Vdd OZ_pq1 300 21596.3 -28800 -12450
p 128 Vdd OZ_pq1 300 21596.3 -27000 -12450
e 329 GND 277 300 3299.75 18600 -6450
e 584 GND 533 300 13497.8 -5550 9000
C IZ_cap2 444 763 (20400 5100)
p 447 584 Vdd 300 6748.75 6150 4200
p 584 Vdd 533 300 13647.8 -5550 4050
p 424 Vdd 453 300 900 27300 1650
p 450 Vdd 424 300 900 16950 1350
p 453 329 589 1200 900 24750 1200
p 447 589 Vdd 1200 900 23250 1200
p 329 Vdd 587 1200 900 21150 1200
e 233 GND 53 300 17996.8 41850 -5550
e 233 GND 53 300 17996.8 40050 -5550
e 233 GND 53 300 17996.8 38250 -5550
e 53 OZ_pq2 GND 300 17996.8 56400 -1350
e 53 OZ_pq2 GND 300 17996.8 58200 -1350
e 53 OZ_pq2 GND 300 17996.8 60000 -1350
e 53 OZ_pq2 GND 300 17996.8 69600 -1350
e 53 OZ_pq2 GND 300 17996.8 71400 -1350
e 53 OZ_pq2 GND 300 17996.8 73200 -1350
p 533 128 Vdd 300 21596.3 -17700 -750
p 533 128 Vdd 300 21596.3 -15900 -750
p 533 128 Vdd 300 21596.3 -14100 -750
p 533 128 Vdd 300 21596.3 -12300 -750
p 533 128 Vdd 300 21596.3 -10500 -750
p IZ_go Vdd 498 300 900 1500 900
p 498 449 Vdd 300 900 2700 900
p 450 Vdd 449 300 900 3900 900
p 449 443 Vdd 600 900 5250 900
p 177 Vdd 444 1200 900 10500 1050
p 584 Vdd 533 300 13647.8 -5550 1200
p 443 Vdd 177 1500 900 7200 1200
p 444 Vdd 450 1200 900 14850 1200
```

```
p 424 587 447 1200 900 19650 1200
e 128 GND OZ_pq1 300 17996.8 -29700 14400
e 128 GND OZ_pq1 300 17996.8 -31500 14400
e 128 GND OZ_pq1 300 17996.8 -33300 14400
e 128 GND OZ_pq1 300 17996.8 -42900 14400
e 128 GND OZ_pq1 300 17996.8 -44700 14400
e 128 GND OZ_pq1 300 17996.8 -46500 14400
e 533 GND 128 300 17996.8 -11550 10800
e 533 GND 128 300 17996.8 -13350 10800
e 533 GND 128 300 17996.8 -15150 10800
e 128 OZ_pq1 GND 300 17996.8 -46500 4050
e 128 OZ_pq1 GND 300 17996.8 -44700 4050
e 128 OZ_pq1 GND 300 17996.8 -42900 4050
e 128 OZ_pq1 GND 300 17996.8 -33300 4050
e 128 OZ_pq1 GND 300 17996.8 -31500 4050
e 128 OZ_pq1 GND 300 17996.8 -29700 4050
e 53 GND OZ_pq2 300 17996.8 56400 9000
e 53 GND OZ_pq2 300 17996.8 58200 9000
e 53 GND OZ_pq2 300 17996.8 60000 9000
e 53 GND OZ_pq2 300 17996.8 69600 9000
e 53 GND OZ_pq2 300 17996.8 71400 9000
e 53 GND OZ_pq2 300 17996.8 73200 9000
e 447 GND 584 300 3299.75 6150 10050
C Vdd GND 17835
C IZ_go GND 1758
C IZ_cap1 GND 1885
C OZ_pq2 GND 11677
C OZ_pq1 GND 11672
C 53 GND 2190
C 128 GND 2176
C 177 GND 271
C 233 GND 528
C 277 GND 142
C 329 GND 114
C 424 GND 84
C 443 GND 51
C 444 GND 245
C 447 GND 109
C 450 GND 59
C 533 GND 518
C 584 GND 152
C IZ_cap2 GND 2220
```

The following is the file produced when the clock
"sim" file is processed by "sim2clip.".

```
(p 128 OZ_pq1 vdd 300 21596.3 -26700 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -28500 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -30300 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -32100 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -33900 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -42900 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -44700 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -46500 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -48300 -24000)
(p 128 OZ_pq1 vdd 300 21596.3 -50100 -24000)
(p 53 OZ_pq2 vdd 300 21596.3 76800 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 75000 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 73200 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 71400 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 69600 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 60600 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 58800 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 57000 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 55200 -29400)
(p 53 OZ_pq2 vdd 300 21596.3 53400 -29400)
(p 277 vdd 233 300 13647.8 24450 -9900)
(p 329 277 vdd 300 6748.75 18600 -11400)
(p 53 vdd OZ_pq2 300 21596.3 53100 -17850)
(p 53 vdd OZ_pq2 300 21596.3 54900 -17850)
(p 53 vdd OZ_pq2 300 21596.3 56700 -17850)
(p 53 vdd OZ_pq2 300 21596.3 58500 -17850)
(p 53 vdd OZ_pq2 300 21596.3 60300 -17850)
(p 53 vdd OZ_pq2 300 21596.3 69300 -17850)
(p 53 vdd OZ_pq2 300 21596.3 71100 -17850)
(p 53 vdd OZ_pq2 300 21596.3 72900 -17850)
(p 53 vdd OZ_pq2 300 21596.3 74700 -17850)
(p 53 vdd OZ_pq2 300 21596.3 76500 -17850)
(p 233 53 vdd 300 21596.3 37200 -17100)
(p 233 53 vdd 300 21596.3 39000 -17100)
(p 233 53 vdd 300 21596.3 40800 -17100)
(p 233 53 vdd 300 21596.3 42600 -17100)
(p 233 53 vdd 300 21596.3 44400 -17100)
(p 277 vdd 233 300 13647.8 24450 -12750)
(n 450 449 520 300 450 3300 -1350)
(n 498 520 gnd 300 450 2700 -1350)
(n IZ_go gnd 498 300 450 1500 -1350)
(n 424 453 gnd 300 450 27600 -1650)
(n 453 gnd 329 1200 450 25500 -2250)
(n 447 329 gnd 1200 450 23550 -2250)
(n 329 gnd 447 1200 450 21600 -2250)
(n 450 424 gnd 300 450 17100 -2700)
```

```
(n 449 443 gnd 600 450 5100 -2700)
(n 424 447 gnd 1200 450 19500 -2850)
(n 177 444 gnd 1200 450 10650 -3150)
(n 444 450 gnd 1200 450 15000 -3300)
(n 443 177 gnd 1200 450 7350 -3300)
(n 277 gnd 233 300 13497.8 24300 -4800)
(p 128 vdd OZ_pq1 300 21596.3 -30600 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -32400 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -34200 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -43200 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -45000 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -46800 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -48600 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -50400 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -28800 -12450)
(p 128 vdd OZ_pq1 300 21596.3 -27000 -12450)
(n 329 gnd 277 300 7289.75 18600 -6450)
(n 284 gnd 533 300 13497.8 -5550 9000)
(p 447 584 vdd 300 6748.75 6150 4200)
(p 584 vdd 533 300 13647.8 -5550 4050)
(p 424 vdd 451 300 600 27300 1650)
(p 450 vdd 424 300 1200 16950 1350)
(n 443 329 589 1200 900 24750 1200)
(p 447 589 vdd 1200 900 23250 1200)
(p 329 vdd 589 1200 900 21750 1200)
(n 233 gnd 53 300 17996.8 41850 -5550)
(n 233 gnd 53 300 17996.8 40050 -5550)
(n 233 gnd 53 300 17996.8 38250 -5550)
(n 53 OZ_pq2 gnd 300 17996.8 56400 -1350)
(n 53 OZ_pq2 gnd 300 17996.8 58200 -1350)
(n 53 OZ_pq2 gnd 300 17996.8 60000 -1350)
(n 53 OZ_pq2 gnd 300 17996.8 69600 -1350)
(n 53 OZ_pq2 gnd 300 17996.8 71400 -1350)
(n 53 OZ_pq2 gnd 300 17996.8 73200 -1350)
(p 533 128 vdd 300 21596.3 -17700 -750)
(p 533 128 vdd 300 21596.3 -15900 -750)
(p 533 128 vdd 300 21596.3 -14100 -750)
(p 533 128 vdd 300 21596.3 -12300 -750)
(p 533 128 vdd 300 21596.3 -10500 -750)
(p IZ_go vdd 498 300 900 1500 900)
(p 498 449 vdd 300 900 2700 900)
(p 450 vdd 449 300 900 3900 900)
(p 449 443 vdd 600 900 5250 900)
(p 177 vdd 444 1200 900 10500 1050)
(p 444 vdd 533 300 13647.8 -5550 1200)
(p 443 vdd 177 1200 900 7200 1200)
(p 444 vdd 450 1200 900 14850 1200)
(p 424 vdd 447 1200 900 19650 1200)
(n 128 gnd OZ_pq1 300 17996.8 -29700 14400)
(n 128 gnd OZ_pq1 300 17996.8 -31500 14400)
```

```
(n 128 gnd OZ_pq1 300 17996.8 -33300 14400)
(n 128 gnd OZ_pq1 300 17996.8 -42900 14400)
(n 128 gnd OZ_pq1 300 17996.8 -44700 14400)
(n 128 gnd OZ_pq1 300 17996.8 -46500 14400)
(n 533 gnd 128 300 17996.8 -11550 10800)
(n 533 gnd 128 300 17996.8 -13350 10800)
(n 533 gnd 128 300 17996.8 -15150 10800)
(n 128 OZ_pq1 gnd 300 17996.8 -46500 4050)
(n 128 OZ_pq1 gnd 300 17996.8 -44700 4050)
(n 128 OZ_pq1 gnd 300 17996.8 -42900 4050)
(n 128 OZ_pq1 gnd 300 17996.8 -33300 4050)
(n 128 OZ_pq1 gnd 300 17996.8 -31500 4050)
(n 128 OZ_pq1 gnd 300 17996.8 -29700 4050)
(n 53 gnd OZ_pq2 300 17996.8 56400 9000)
(n 53 gnd OZ_pq2 300 17996.8 58200 9000)
(n 53 gnd OZ_pq2 300 17996.8 60000 9000)
(n 53 gnd OZ_pq2 300 17996.8 69600 9000)
(n 53 gnd OZ_pq2 300 17996.8 71400 9000)
(n 53 gnd OZ_pq2 300 17996.8 73200 9000)
(n 447 gnd 584 300 3299.75 6150 10050)
```

Appendix D: User's Manual

Extraction Program

The extraction program can be run on any machine that can run CLIPS. Machines with more memory and that are more powerful are preferable.

Files needed are:

clips
sim2clip
"sim"
tr1.clp
tr2.clp
tr3.clp
findext.clp
trn.btt
findext.btt

The steps to perform the extraction are:

1. Rename the "sim" file to "new.sim". This "sim" file is the output of the program MEXTRA.
2. Run the program sim2clip. A new file called good.clp should now be in the directory. This file is the processed "new.sim" file.
3. Run the extraction routines by typing "clips -f trn.btt". This calls up CLIPS and runs the batch file trn.btt, which contains the commands needed to run the various rule files. The output of this process are four files; outcomp1.clp, comprem1.clp, outcomp2.clp and comprem2.clp. This files contain the first and second levels of extraction.

4. Run the routines to find the extreme values, and to produce the input file for the display routines, by typing the command "clips -f findext.btt". This calls up CLIPS and runs the batch file findext.btt. At the end of this step there should be a file called "scaled.clp" in the directory. Rename this file before further extraction on any other circuits, or it will be overwritten.

Display Program

The display program can be run on any Sun 3 or Sun 4 workstation, provided that the code has been correctly compiled for it. The program must be run from inside the Sunview environment. The environment is started by typing "sunview".

Files needed are :

```
drawgate
drawit
scaled.clp (or what it was renamed)
```

The steps to use the graphical display are as follows:

1. Enter Sunview by typing "sunview".
2. Start the graphical display by typing "drawgate". Wait for the new window to open.
3. Place the mouse pointer in the window, then type in the name of the file produced in the extraction process, then press RTN. Unless it was renamed, the file is

"scaled.clp". Do not use quotes when entering the name, or the program will not be able to find the file.

4. Enter the minimum and maximum x and y coordinates as asked by the program.

5. To escape the display, push the right mouse button while the mouse pointer is on the graphical display. The input window will return and ask if another file or view is wanted. Answer y or n. Any answer besides y will cause the program to exit. If y is entered, return to step 3.

Bibliography

1. Fretheim, CPT Erik J. Reverse Engineering VLSI Using Pattern Recognition Techniques. *MS Thesis AFIT/GE/ENG/88J-1*. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, June 1988.

2. Dukes, CPT Michael A. A Multiple-Valued Logic System for Circuit Extraction to VHDL 1076-1987. *MS Thesis AFIT/GE/ENG/88S-1*. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, September 1988.

3. NASA, Lyndon B. Johnson Space Center, Artificial Intelligence Section. *CLIPS Reference Manual, Version 4.3 of CLIPS*. June 1989.

4. Bratko, Ivan. *PROLOG Programming for Artificial Intelligence*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1987.

5. Kernighan, Brian W. and Dennis M. Ritchie. *The C Programming Language(Second Edition)*. Englewood Falls NJ: Prentice Hall, 1988.

6. -----. *SunCore Reference Manual*. Sun Microsystems, Mountain View, CA, 1986.

Vita

Captain Stuart A. Yarost was born in Detroit, Michigan on 17 February 1963. Following graduation from high school in Southfield, Michigan in 1981, he attended Michigan State University, where he graduated with honor with a Bachelor of Science in Electrical Engineering. During his time at Michigan State University, he entered the College Senior Engineering Program (CSEP) of the USAF. This led to his commission in the USAF after finishing Officer Training School on 13 September 1985. His first assignment was to the 6520th Test Group at Edwards AFB, where his jobs included microcomputer manager, training manager, program analyst, and avionics flight test engineer at the F-16 Combined Test Force. He was selected for full-time attendance to the Air Force Institute of Technology in December of 1987. He is happily married to his wife Debbie, and will remain at Wright-Patterson AFB after graduation, to work for AFLC at ALD.

Permanent address: 7080 Clements

West Bloomfield, Mi., 48322

19. (cont) ABSTRACT

This thesis proposes a system for higher-order logic extraction of components from a net-list of transistors and the graphical display of the extracted components. Critical sections have been implemented to demonstrate the feasibility of the system. These sections include a prototype expert system written in CLIPS and a graphical display capable of displaying extracted components on a Sun workstation.

Extraction techniques which were developed in this effort use pattern matching and multiple passes. Graphical techniques used in the display include simple line drawing and translation of images.

This research has the potential to provide savings of time and effort to engineers designing new circuits or reverse-engineering older circuits for which no adequate specifications exist. This system will also help to close the design cycle and allow the designer to assure that what he has physically designed is what he has logically designed.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCE/ENG/89D-9	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology(AU) Wright-Patterson AFB, OHIO 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION WRDC/ELED 8b. OFFICE SYMBOL (If applicable) WRDC/ELED	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Wright Research and Developement Center Wright-Patterson AFB, OHIO 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
(UNCLASSIFIED)
A CIRCUIT EXTRACTION SYSTEM AND GRAPHICAL DISPLAY FOR VLSI DESIGN

12. PERSONAL AUTHOR(S)
Stuart A. Yarost, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1989 December	122

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Circuit Extraction, Reverse Engineering, Integrated Circuits Computer Aided Design, Artificial Intelligence, Graphical Display
09	01		
12	09		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Dr. Frank M. Brown, Professor of Electrical Engineering

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Frank M. Brown	22b. TELEPHONE (Include Area Code) (513)255-9265 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

END

DATE

FILMED

1-90

DTIC